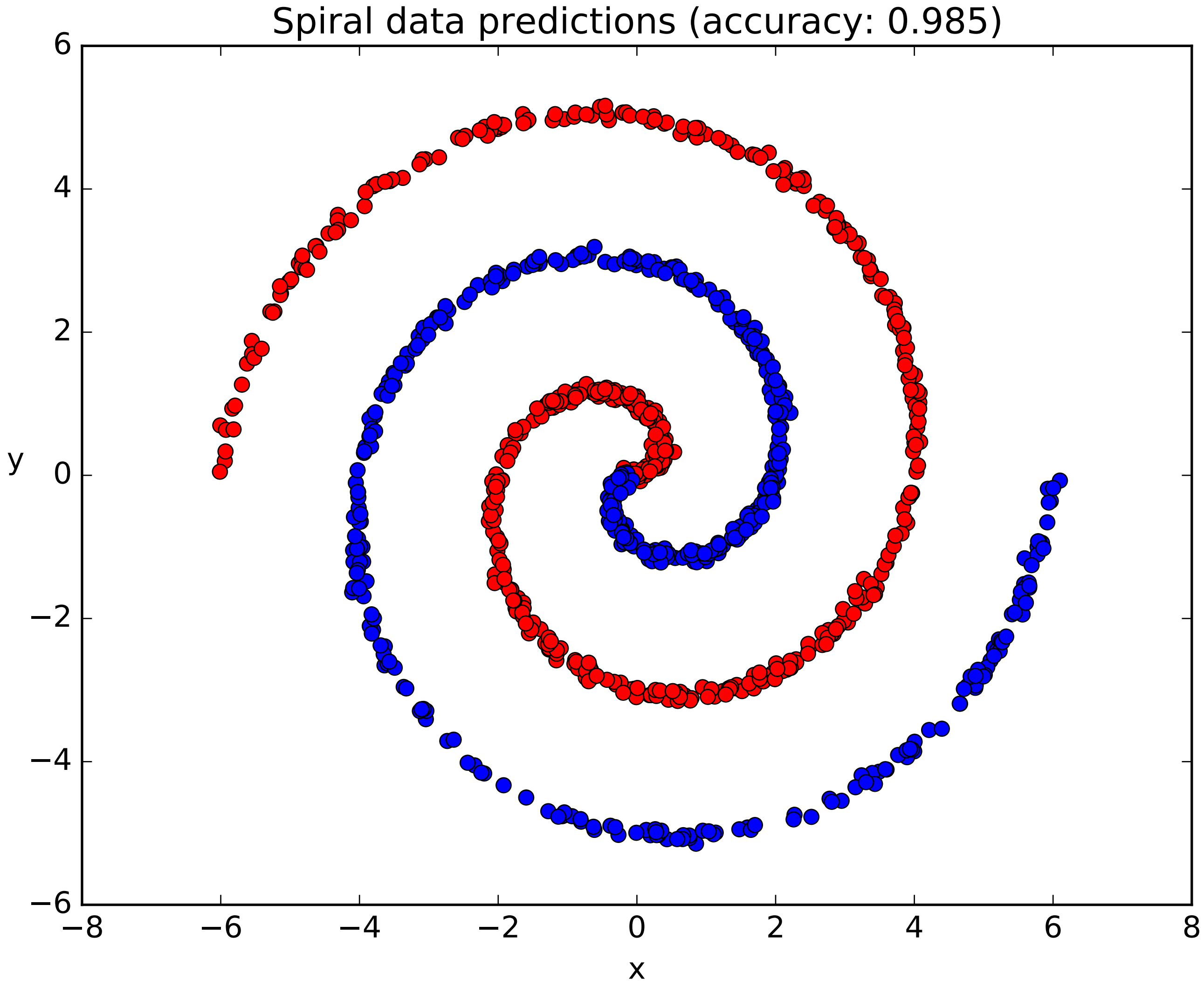
Assignment 2 - Spirals

Caleb Zulawski

The implemented perceptron takes 2 inputs (x, y) and returns two outputs (p(t=0), p(t=1)). It has 2 hidden layers with 20 neurons each. It is able to classify with near perfect accuracy; classifying the data points near the center of the spiral is nearly impossible since the noise hides any information about which arm the data is from.


Spiral data predictions (accuracy: 0.985)
6
4
2
0
−2
−4
−6
y
−8
−6
−4
−2
0
2
4
6
8
x

```python
#!/usr/bin/env python3

# Caleb Zulawski
# ECE411 - Computational Graphs for Machine Learning
# Assignment 2 - Binary classification on a spiral dataset

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

def get_data(N):
    a = 2;
    t = np.random.uniform(-3, 3, N)
    c = t > 0
    sign = np.where(c, 1, -1)
    x = a*t*np.cos(np.pi*t)
    y = a*t*np.sin(np.pi*t) * sign
    x += np.random.normal(0, 0.05, N);
    y += np.random.normal(0, 0.05, N);
    return x, y, c;

class Model():
    def __init__(self, sess, n_epochs, learning_rate):
        self.sess = sess
        self.n_epochs = n_epochs
        self.learning_rate = learning_rate
        self.build_model()

    def build_model(self):
        n_hidden_1 = 20
        n_hidden_2 = 20
        n_input = 2
        n_output = 2

        self.x = tf.placeholder(tf.float32, [n_input, 1])
        self.y = tf.placeholder(tf.float32, [1, n_output])

        w1 = tf.get_variable(shape=[n_input, n_hidden_1], dtype=tf.float32, name="weight1", initializer=tf.random_normal_initializer())
        w2 = tf.get_variable(shape=[n_hidden_1, n_hidden_2], dtype=tf.float32, name="weight2", initializer=tf.random_normal_initializer())
        wo = tf.get_variable(shape=[n_hidden_2, n_output], dtype=tf.float32, name="weighto", initializer=tf.random_normal_initializer())

        b1 = tf.get_variable(shape=[n_hidden_1], dtype=tf.float32, name="bias1", initializer=tf.random_normal_initializer())
        b2 = tf.get_variable(shape=[n_hidden_2], dtype=tf.float32, name="bias2", initializer=tf.random_normal_initializer())
        bo = tf.get_variable(shape=[n_output], dtype=tf.float32, name="biaso", initializer=tf.random_normal_initializer())

        layer_1 = tf.add(tf.matmul(tf.transpose(self.x), w1), b1)
        layer_1 = tf.nn.relu(layer_1)
        layer_2 = tf.add(tf.matmul(layer_1, w2), b2)
        layer_2 = tf.nn.relu(layer_2)

        self.layer_out = tf.matmul(layer_2, wo) + bo
        self.out = tf.nn.softmax(self.layer_out)

    def train(self, xs, ys, cs):
        cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=self.layer_out, labels=self.y))
        vs = tf.trainable_variables()
        cost += tf.add_n([ tf.nn.l2_loss(v) for v in vs if ’bias’ not in v.name ]) * 0.001
        optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_rate).minimize(cost)
        self.sess.run(tf.global_variables_initializer())
        for epoch in range(self.n_epochs):
            for x, y, c in zip(xs, ys, cs):
                v = np.expand_dims(np.asarray([x, y]), axis=1)
                l = np.reshape([not(c).astype(float), c.astype(float)], (1, 2))
                self.sess.run(optimizer, feed_dict={self.x: v, self.y: l})

    def predict(self, x, y):
        v = np.expand_dims(np.asarray([x, y]), axis=1)
        return self.sess.run(self.out, feed_dict={self.x: v})[0][1] > 0.5

n_train = 1000; # if the training set is sparser, sometimes there are gaps in the spiral and it doesn’t train well
n_test = 1000;
x_train, y_train, c_train = get_data(n_train)
x_test, y_test, c_test = get_data(n_test)

with tf.Session() as sess:
    model = Model(sess, 200, 0.01)
    model.train(x_train, y_train, c_train)

    total = 0;
    pred = np.array([]);
    for x, y, c in zip(x_test, y_test, c_test):
        pred = np.append(pred, model.predict(x, y))

    plt.figure()
    plt.xlabel(’x’)
    plt.ylabel(’y’, rotation=0)
    plt.title(’Spiral data predictions (accuracy: {})’.format(np.sum(np.equal(c_test, pred))/n_test))
    plt.plot(x_test[pred == True], y_test[pred == True], ’ro’)
    plt.plot(x_test[pred == False], y_test[pred == False], ’bo’)
    plt.show()
```