# ECE472 - Deep Learning: Assignment 2

*Authors*
Alexander KOLDY

September 15 2021
ECE472 - Fall 2021

# Attempts

Data parameters (i.e., batch size), training parameters (i.e., number of epochs and learning rate) and neural network parameters (i.e., layer depth and layer width) were tuned in order find convergence in the neural network's loss function.

In terms of data parameters, the batch size was selected to be 70, where the full training data-set was 600 points. This batch size is approximately 11% of the full data-set. Earlier, much lower batch sizes were chosen (i.e., 10), which produced faster convergence rates, but caused the neural network to make inaccurate predictions on the full training data-set. In other words, low batch sizes caused the neural network to predict poorly, even with a very low loss.

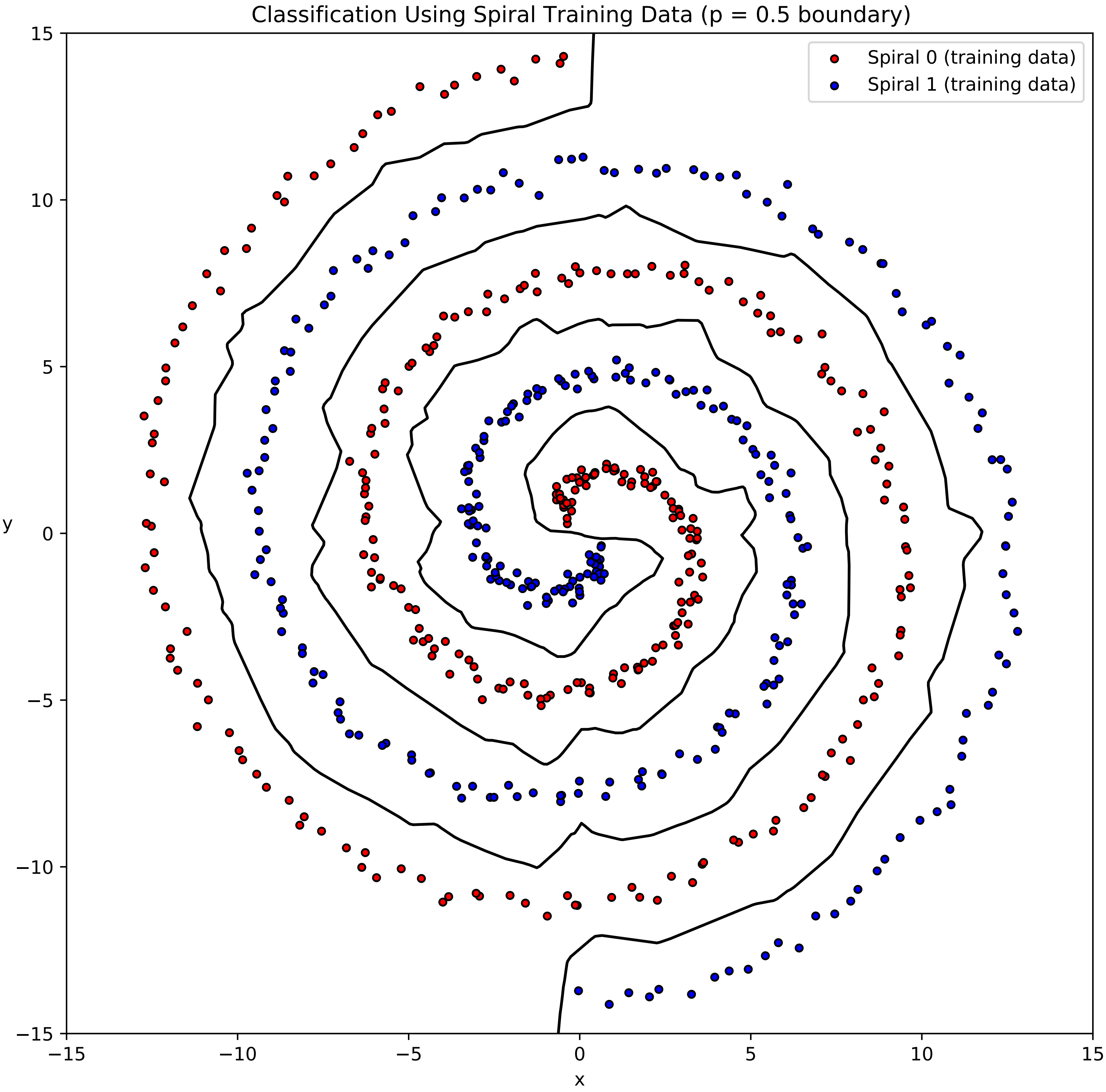
In terms of the neural network itself, a hidden layer depth of 1 was originally chosen with a width of 2000. The loss had a difficult time decreasing, so the other extremity was tested: a neural network of depth 10 and widths of 20 was constructed. This neural network also had trouble decreasing the loss. After reconstructing the network many more times, a depth of 2 with widths of 60 and 25, respectively, were chosen. This network showed signs of convergence, however it was very slow to converge. Thus, this was compensated by changing the training parameters. Approximately 20,000 epochs with a learning rate of 0.03 yielded a convergence, where the network predicted the entire data-set with 100% accuracy. This was later changed to an even higher extremity of 110,000 in order to ensure a high-quality graph was produced.

It is important to note that these parameters are not the best, and further investigation could potentially lead to much faster convergences. However, due to limited time, finding these parameters is not possible.

```python
'''
Alexander Koldy
ECE 472 - Deep Learning
Assignment 2
'''

'''
Perform binary classification on the spirals dataset
using a multi-layer perceptron.You must generate
the data yourself.
'''

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.python.ops.array_ops import meshgrid

tf.enable_eager_execution()

'''
Generates noisy spirals for training data
'''
class Spiral():
  def __init__(self, theta, direction=np.array([1, 1])):
    self.x = direction[0]*theta*np.cos(theta)
    self.y = direction[1]*theta*np.sin(theta)

  def add_noise(self, sigma_noise, N):
    epsilon = np.random.normal(0, sigma_noise, N)
    self.x += epsilon
    epsilon = np.random.normal(0, sigma_noise, N)
    self.y += epsilon

'''
Collects data from noisy spirals.
Has shuffling functionality, and
is able to create batches for
training
'''
class Data():
  def __init__(self, is_training_data):
    self.N = 300

    if is_training_data:
      '''Training data organization parameters'''
      self.batch_size = 70

      '''Training data generation parameters'''
      self.sigma_noise = 0.2
      self.theta = np.linspace((1/4)*np.pi, 4.5*np.pi, self.N)
      self.direction_0 = np.array([-1, 1])
      self.direction_1 = np.array([1, -1])

      '''Generate spirals'''
      self.spiral_0 = Spiral(self.theta, self.direction_0)
      self.spiral_0.add_noise(self.sigma_noise, self.N)
      self.spiral_1 = Spiral(self.theta, self.direction_1)
      self.spiral_1.add_noise(self.sigma_noise, self.N)
      spiral_x = np.concatenate((self.spiral_0.x, self.spiral_1.x), axis=0)
      spiral_y = np.concatenate((self.spiral_0.y, self.spiral_1.y), axis=0)

      '''Input/output data for training'''
      self.X = np.vstack((spiral_x, spiral_y))
      self.Y = np.concatenate((np.zeros((self.N, )), np.ones((self.N, ))), axis=0)
    else:
      '''Testing data generation'''
      x = np.linspace(-15, 15, self.N)
      y = np.linspace(-15, 15, self.N)
      self.x, self.y = np.meshgrid(x, y)

      '''Input data for testing'''
      self.X = np.vstack((self.x.flatten(), self.y.flatten()))
      self.Y = np.zeros((1, 1))

  def shuffle(self):
    new_order = np.random.permutation(2*self.N)
    self.X = self.X[:, new_order]
    self.Y = self.Y[new_order]

  def get_batch(self, batch_size ):
    batch_start = np.random.randint(0, 2*self.N - batch_size - 1)
    batch_end = batch_start + batch_size
    X_batch = tf.convert_to_tensor(self.X[:, batch_start:batch_end].T, dtype=tf.float64)
    Y_batch = tf.convert_to_tensor(self.Y[batch_start:batch_end], dtype=tf.float64)

    return X_batch, Y_batch

  def convert_data(self):
      self.X = tf.convert_to_tensor(self.X.T)
      self.Y = tf.convert_to_tensor(self.Y)

'''
Multi-layer perceptron which contains nested class,
Layer. Constructs neural network to solve
binary classification problem.
'''
class Neural_Network(tf.Module):
  '''
  Layer of perceptrons which can
  be generated more than once
  (i.e., multiple hidden layers)
  '''
  class Layer(tf.Module):
    def __init__(self, activation_type, num_inputs, width):
      self.activation_type = activation_type
      self.width = width

      '''Perceptron parameters'''
      self.W = tf.Variable(tf.random.normal(stddev=1, shape=(num_inputs, width), dtype=tf.float64))
      self.b = tf.Variable(tf.random.normal(stddev=1, shape=(1, width), dtype=tf.float64))

    def update(self, input):
      def activation(t):
        if self.activation_type == 'ReLu':
          return tf.nn.relu(t)
    elif self.activation_type == 'Sigmoid':
     return tf.nn.sigmoid(t)
    else:
     return t
   return activation(input @ self.W + self.b)

 '''
 Establish a neural network with:
 depth: number of hidden layers
 width: list of size depth containing
 desired width of each layer
 data: all training data available
 '''
 def __init__(self, depth, widths, data):
  '''Data'''
  self.data = data

  '''Learning parameters'''
  self.epochs = 110000
  self.learning_rate = 0.03

  '''Create neural network'''
  self.layers = []
  self.create_layer('ReLu', 2, widths[0])
  for i in range(1, depth):
   self.create_layer('ReLu', widths[i - 1], widths[i])
  self.create_layer('None', self.layers[-1].width, 1)

 def create_layer(self, activation_type, num_inputs, width):
  layer = self.Layer(activation_type, num_inputs, width)
  self.layers.append(layer)

 def train(self):
  optimizer = tf.keras.optimizers.SGD(learning_rate=self.learning_rate)

  for epoch in range(self.epochs):
   X, Y = self.data.get_batch(self.data.batch_size)
   with tf.GradientTape() as tape:
    input = X
    for layer in self.layers:
     output = layer.update(input)
     input = output
    y_hat = output
    y_hat = tf.reshape(tf.convert_to_tensor(y_hat, dtype=tf.float64),
(self.data.batch_size, ))

    loss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=Y,
logits=y_hat))

   gradients = tape.gradient(loss, self.trainable_variables)

   print(f"Epoch count {epoch}: Loss value: {loss.numpy()}")

   optimizer.apply_gradients(zip(gradients, self.trainable_variables))

 def deploy(self, data, is_training_data):
  num_correct = 0
  num_incorrect = 0

```

```
    data.convert_data()
    X, Y = data.X, data.Y

    input = X
    for layer in self.layers:
      output = layer.update(input)
      input = output
    output = tf.nn.sigmoid(output)
    y_hat = output.numpy().flatten()

    if is_training_data:
      y = Y.numpy()
      y_hat = np.round(y_hat)

      for i in range(2*self.data.N):
        print("y: " + str(y[i]) + " | y_hat: " + str(y_hat[i]))

        if y_hat[i] == y[i]:
          num_correct += 1
        else:
          num_incorrect += 1

      print("Number correct: " + str(num_correct))
      print("Number incorrect: " + str(num_incorrect))

      return num_incorrect
    else:
      return y_hat

training_data = Data(True)
testing_data = Data(False)
nn = Neural_Network(2, [60, 25], training_data)

'''Shuffle data before starting'''
nn.data.shuffle()

'''Train data and deploy on full training dataset'''
nn.train()
nn.deploy(training_data, True)

'''Deploy to meshgrid'''
output = nn.deploy(testing_data, False)

'''Plot Results'''
data = Data(True)
plt.figure(figsize=(10, 10))
plt.title('Classification Using Spiral Training Data (p = 0.5 boundary)')
plt.xlabel("x")
plt.ylabel("y", rotation="horizontal")
plt.scatter(data.spiral_0.x, data.spiral_0.y, color='red', edgecolors='black', s=15, zorder=2, label='Spiral 0 (training data)')
plt.scatter(data.spiral_1.x, data.spiral_1.y, color='blue', edgecolors='black', s=15, zorder=2, label='Spiral 1 (training data)')
plt.contour(testing_data.x, testing_data.y, np.reshape(output, testing_data.x.shape), levels=1, colors='black')
plt.legend()
plt.show()

'''
References:
```

```
(1) Cooper Union ECE-472: Deep Learning - Learning Materials
(2) https://www.codegrepper.com/code-examples/python/draw+spiral+in+matplotlib
(3) https://towardsdatascience.com/building-neural-network-from-scratch-9c88535bf8e9
(4) https://stackoverflow.com/questions/4601373/better-way-to-shuffle-two-numpy-arrays-in-unison
(5) https://gist.github.com/ccurro/822bff081babc4a979375e59bce7d981
(6) https://machinelearningknowledge.ai/matplotlib-contour-plot-tutorial-for-beginners/
(7) https://github.com/yuvalofek/Deep-Learning
'''
```


Classification Using Spiral Training Data (p = 0.5 boundary)
Spiral 0 (training data)
Spiral 1 (training data)
15
10
5
0
−5
−10
−15
y
−15
−10
−5
0
5
10
15
x