Deep Learning Course (980)

Assignment One

Assignment Goals:

- Start with TensorFlow (version 1.0).
- Implement and apply a multi-layer feed-forward neural network classifier.
- Understand the differences and trade-offs between linear regression, logistic regression, and multi-layer feed-forward neural network.

In this assignment, you will be asked to install TensorFlow and Jupyter Notebook. In addition, you will design a Multilayer feed-forward neural network to classify a Toy Dataset (Figure 1).

**DataSet:** dataset has 100 instances and two features.

1. Install TensorFlow (1.15.0) and Jupyter Notebook. (15 points) Run the provided code [Linear Regression](http://cs229.stanford.edu/notes/cs229-notes1.pdf). This code uses linear regression and threshold classifier to classify the TOY Dataset. Analyze the classifier result. The implemented regression code has a problem compared to the linear regression model. Correct the code and describe the effect on classification accuracy. What is the reason for the change in accuracy? (10 points)

2. Using code similar to what was provided for linear regression, implement logistic regression. How does the result change. Hint: What is the correct loss function for logistic regression compared to linear regression? (20 points)

3. Implement a multi-layer feed-forward neural net and try to reach 100 accuracy (You are not allowed to use Keras and tf.nn and tf.losses for this part). Compare regression model's capacity with the multi layer. (20 points)

4. Use tf.keras to implement the exact graph that you implemented in the section 3. (25 points)

**Submission Notes:**

Please use Jupyter Notebook. The notebook should include the final code, results and your answers. You should submit your Notebook in .pdf and .ipynb format. (10 points)

You can use visualize() helper function to visualize the model's decision boundary.

**Instructions:**

The university policy on academic dishonesty and plagiarism (cheating) will be taken very seriously in this course. Everything submitted should be your own writing or coding. You must not let other students copy your work. Spelling and grammar count.

Your assignments will be marked based on correctness, originality (the implementations and ideas are from yourself), clarification and test performance.
In [1]:
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

#this line makes the notebook put the figures in-line rather than generate them in new windows
In [2]: # helper functions

# helper function for generating the data
def data_generator(N=100, D=2, K=2):
  # N number of points per class; D dimensionality; k number of classes
  np.random.seed(0)
  X = np.zeros((N*K,D))
  y = np.zeros((N*K), dtype='uint8')
  for j in range(K):
    ix = range(N*j,N*(j+1))
    r = np.linspace(0.0,1,N) # radius
    t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta
    X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
    y[ix] = j
  fig = plt.figure()
  plt.title('Figure 1: DataSet')
  plt.scatter(X[:,0], X[:,1], c=y, s=40, cmap=plt.cm.Spectral)

  plt.xlim(X.min()-0.5, X.max()+0.5)
  plt.ylim(X.min()-0.5, X.max()+0.5)
  return X, y

# helper function for visualizing the boundaries
def visualize(sample, target, predict, se):
  """
  function for visualizing the classifier boundaries on the TOY data set.

  @param sample: Training data features
  @param target: Target
  @param predict: Model prediction
  @param se: The model's session
  """
  h = 0.02
  x_min, x_max = sample[:,0].min() - 1, sample[:,0].max() + 1
  y_min, y_max = sample[:,1].min() - 1, sample[:,1].max() + 1
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                        np.arange(y_min, y_max, h))
  Z = np.round(se.run(predict,{X:(np.c_[xx.ravel(), yy.ravel()])}))
  Z = Z.reshape(xx.shape)
  fig = plt.figure()
  plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)
  plt.scatter(sample[:,0], sample[:,1], c=target, s=40, cmap=plt.cm.Spectral)
  plt.xlim(xx.min(), xx.max())
  plt.ylim(yy.min(), yy.max())
In [3]:

    # TOY DataSet
    sample, target = data_generator()
    # print(target.shape)

    # Figure 1: DataSet
    
    ![Figure 1: DataSet](image)

    
    In [6]:

    tf.set_random_seed(1)

    # Almost-correct Linear Regression
    X = tf.placeholder(tf.float32, [None, 2])
    Y = tf.placeholder(tf.float32, [None, 1])

    W = tf.Variable(tf.random_normal(shape=[2, 1], seed=1))

    first_layer = (tf.matmul(X, W))
    objective_function = tf.reduce_mean((tf.square(first_layer - Y)))
    # the reduce_sum seems to operate on just 1 element

    LR = tf.train.GradientDescentOptimizer(learning_rate=.1).minimize(objective_function)

    # predicted value above 0.5 -> predict = 1 = classify as positive
    predict = tf.cast(tf.greater(first_layer, .5), tf.float32)

    accu = tf.reduce_mean(tf.cast(tf.equal(predict, Y), tf.float32))

    se = tf.Session()
    se.run(tf.global_variables_initializer())

    for i in range(100):
        se.run(LR, {X: sample, Y: target.reshape(-1, 1)})
        # -1 is like "unspecified"

        print("Epoch:", (i + 1), "loss: ", "{:.3f}".format(se.run(objective_function, {X: sample, Y: target.reshape(-1, 1)})), "acc: ", "{:.3f}".format(se.run(accu, {X: sample, Y: target.reshape(-1, 1)})))

    visualize(sample, target, predict, se)