Assignment 1

January 17, 2021

1 Deep Learning Course

1.1 Assignment 1

Assignment Goals:

- Start with Keras.
- Implement and apply logistic regression and multi-layer feed-forward neural network classifiers.
- 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 Keras and Jupyter Notebook. Keras is packaged with TensorFlow 2.0 as tensorflow.keras, so you actually need to install Tensorflow 2.0. (TA's environment to run your code is Python 3.7 + Tensorflow 2.4). In addition, you are required to design several models to classify a Toy Dataset (Figure 1).

Dataset: We provide a toy dataset, which has 200 instances and 2 features. See below "Toy Data and Helper Functions" section for toy data generation code.

You do not need to generate separated training dataset and test dataset for this assignment. Both training and prediction will both be on one dataset. Directly use the "sample, target" variables we provide as the dataset for your assignment.

In the following accuracy is defined as the empirical accuracy on the training set, that is, accuracy = {number of correctly predicted instances}/{number of all instances in dataset}).

Requirements

1. Install Keras (Tensorflow 2) and Jupyter Notebook. (10 points)
2. Implement a logistic regression to classify the Toy Dataset. (20 points) We have provided a very simple linear regression example, please refer to the example and implement your logistic regression model.
    - You should determine: what loss function and what optimization algorithm do you plan to use? (4 points)
    - Try to reach $> 72\%$ accuracy. (4 points)
    - We have provided a visualize() helper function, you can visualize the model's decision boundary using that function. What's more, you are asked to compute and visualize **the equation of the decision boundary** of your trained **logistic regression**. Fill in the 'equation of decision boundary' column in the following table. Then you can modify

the visualize() function or implement a new visualization function to draw the linear decision boundary (Hint: should be a straight line aligned with the decision boundary plotted in visualize()). (5 points)

3. Implement a multi-layer linear neural network (>= 2 hidden layers) to classify the Toy Dataset. (20 points) A deep linear neural network is a deep feed-forward neural network without activation functions (See here, page 11-13 for detail introduction of linear neural networks).

   - You should determine: what loss function and what optimization algorithm do you plan to use, what is your network structure? (4 points)
   - Try to reach $> 72\%$ accuracy. (4 points)
   - Compute and visualize **the equation of the decision boundary** of your trained **linear neural network**. Fill in the 'equation of decision boundary' column in the following table. Then you can modify the visualize() function or implement a new visualization function to draw the linear decision boundary. (5 points)

4. Implement a multi-layer feed-forward neural network (>= 2 hidden layers). (20 points)

   - You should determine: what loss function and what optimization algorithm do you plan to use? what is your network structure? what activation function do you use? (5 points)
   - Try to reach 100% accuracy. (5 points)

5. Add L2-regularization to your implemented nonlinear neural network in (4.). Set the coefficient of L2-regularization to be 0.01, 2, 100, respectively. How do different values of coefficient of L2-regularization affect the model (i.e., model parameters, loss value, accuracy, decision boundary)? You can use a table to compare models trained without regularization, with different coefficients of regularization. (20 points)

   - Please draw your table and analysis in the '**Answers and Analysis**' section.

You should:

- Train each of your models to its best accuracy. Then fill in the following table in the '**Answers and Analysis**' section.
- Complete the '**Answers and Analysis**' section.

Answers and Analysis

- First, fill in the following table. The '-' indicates a cell that does not need to be filled in.

Model	Loss	Accuracy	Equation of Decision Boundary	NN Structure	Activation Function	Loss Function
Linear Regression	0.14	72.96%	$-1.28x_1 + 7.62x_2 - 0.07 = 0$	-	-	Mean Square Error
Logistic Regression				-	-	
Linear Neural Network						

Model	Loss	Accuracy	Equation of Decision Boundary	NN Structure	Activation Function	Loss Function
Feedforward Neural Network			-			

- Then, compare and analyze the classification results of your models. In particular, are there any differences between the performance (i.e., accuracy, loss value) of linear regression, logistic regression, linear neural network and deep nonlinear neural network? What do you think is the reason for the difference? (10 points)
- Your table and analysis of (5. Add L2-regularization) here.

Submission Notes: Please use Jupyter Notebook. The notebook should include the final code, results and your answers. You should submit your Notebook in both .pdf and .ipynb format.

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), clarity and performance. Clarity means whether the logic of your code is easy to follow. This includes 1) comments to explain the logic of your code 2) meaningful variable names. Performance includes loss value and accuracy after training.

1.2 Your Implementation

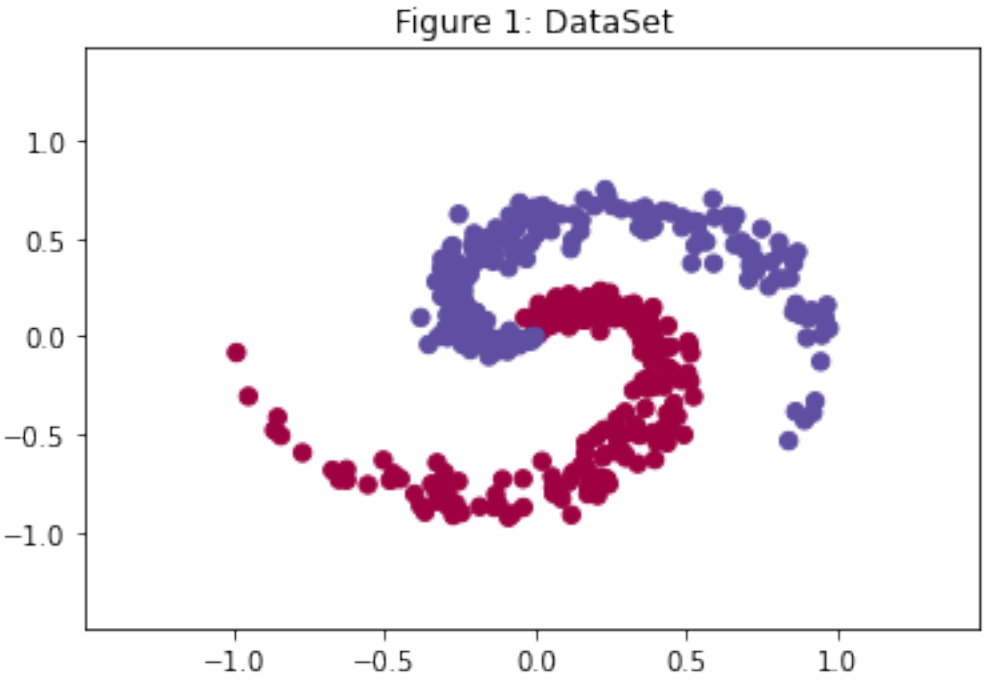
1.2.1 Toy Data and Helper Functions

[1]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
from keras.models import Sequential
import matplotlib.pyplot as plt
```

[2]:
```
# helper functions

# helper function for geterating the data
def data_generator(N = 200,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)*0.3 # 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()-.5, X.max()+.5)
    plt.ylim(X.min()-.5, X.max()+.5)

    return X,y


# helper function for visualizing the decision boundaries
def visualize(sample, target, model):
    """
    function for visualizing the classifier boundaries on the TOY dataset.

    sample: Training data features
    target: Target
    model: the model
    """
    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 = model.predict_classes(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())

    return
```

[3]:
```
#  TOY DataSet
sample, target = data_generator(N = 200)
# print(target.shape)
```



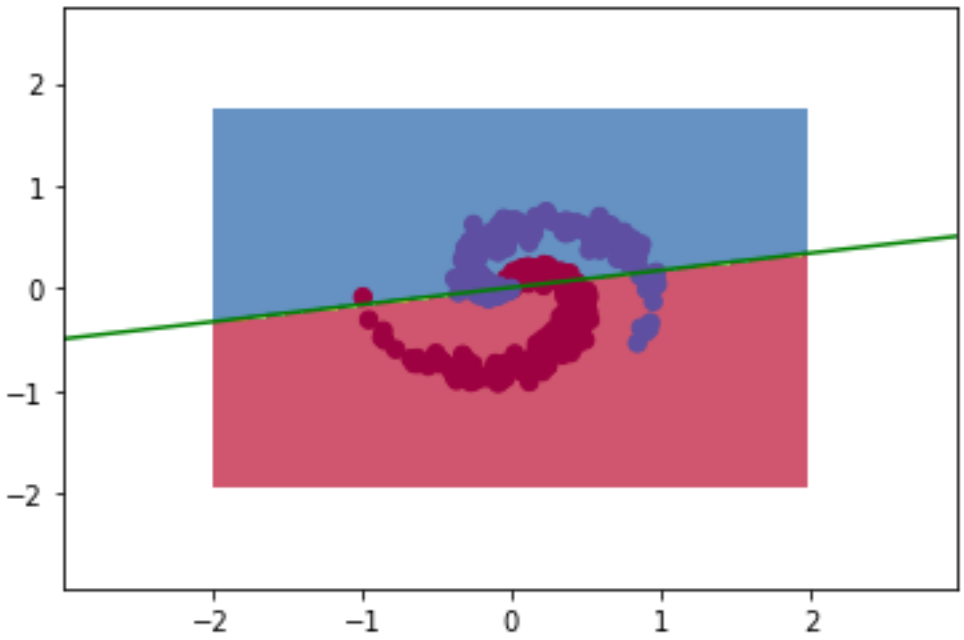
1.2.2 Given Example: Linear Regression

Note that linear regression is usually used for regression tasks, not classification tasks. However, it can be used for binary classification problems (be labeled 0, 1) with a threshold classifier. That is, when linear regression outputs > 0.5, the prediction is 1; otherwise, the prediction is 0.

[ ]:
```
#Define the model
linear_regression = keras.Sequential()

linear_regression.add(keras.layers.Dense(1,"linear"))

linear_regression.compile(optimizer="sgd",loss= keras.losses.
 ↪MeanSquaredError(), metrics=['accuracy'])

# Train the model
value_epoches = linear_regression.fit(sample, target, epochs=5,batch_size=1)
```

[ ]:
```
visualize(sample,target, linear_regression)
```

Here is an example: the green line is the line of the decision boundary. You should draw the linear decision boundary like this.



1.2.3 Logistic Regression

[ ]:
```
# implement your logistic regression here
```

1.2.4 Deep Linear Neural Network

[ ]:
```
# # implement your nonlinear feed forward neural network here
```

1.2.5 Deep Neural Network

[ ]:
```
# implement your nonlinear feed forward neural network here
```

1.2.6 Deep Neural Network with L2-regularization

$\lambda = 0.01, 2, 100$

[ ]:
```

```