exercise3

June 8, 2020

1 Programming Exercise 3

2 Multi-class Classification and Neural Networks

2.1 Introduction

In this exercise, you will implement one-vs-all logistic regression and neural networks to recognize handwritten digits. Before starting the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

All the information you need for solving this assignment is in this notebook, and all the code you will be implementing will take place within this notebook. The assignment can be promptly submitted to the coursera grader directly from this notebook (code and instructions are included below).

Before we begin with the exercises, we need to import all libraries required for this programming exercise. Throughout the course, we will be using `numpy` for all arrays and matrix operations, `matplotlib` for plotting, and `scipy` for scientific and numerical computation functions and tools. You can find instructions on how to install required libraries in the README file in the github repository.

In [1]:
```
# used for manipulating directory paths
import os

# Scientific and vector computation for python
import numpy as np

# Plotting library
from matplotlib import pyplot

# Optimization module in scipy
from scipy import optimize

# will be used to load MATLAB mat datafile format
from scipy.io import loadmat

# library written for this exercise providing additional functions for assignment subm
import utils

# define the submission/grader object for this exercise
```

```
grader = utils.Grader()

# tells matplotlib to embed plots within the notebook
%matplotlib inline
```

2.2 Submission and Grading

After completing each part of the assignment, be sure to submit your solutions to the grader. The following is a breakdown of how each part of this exercise is scored.

	Section	Part	Submission function	Points
1		Section **??**	Section **??**	30
2		Section **??**	Section **??**	20
3		Section **??**	Section **??**	20
4		Section **??**	Section **??**	30
		Total Points		100

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration.

At the end of each section in this notebook, we have a cell which contains code for submitting the solutions thus far to the grader. Execute the cell to see your score up to the current section. For all your work to be submitted properly, you must execute those cells at least once. They must also be re-executed everytime the submitted function is updated.

2.3 1 Multi-class Classification

For this exercise, you will use logistic regression and neural networks to recognize handwritten digits (from 0 to 9). Automated handwritten digit recognition is widely used today - from recognizing zip codes (postal codes) on mail envelopes to recognizing amounts written on bank checks. This exercise will show you how the methods you have learned can be used for this classification task.

In the first part of the exercise, you will extend your previous implementation of logistic regression and apply it to one-vs-all classification.

2.3.1 1.1 Dataset

You are given a data set in `ex3data1.mat` that contains 5000 training examples of handwritten digits (This is a subset of the MNIST handwritten digit dataset). The `.mat` format means that that the data has been saved in a native Octave/MATLAB matrix format, instead of a text (ASCII) format like a csv-file. We use the `.mat` format here because this is the dataset provided in the MATLAB version of this assignment. Fortunately, python provides mechanisms to load MATLAB native format using the `loadmat` function within the `scipy.io` module. This function returns a python dictionary with keys containing the variable names within the `.mat` file.

There are 5000 training examples in `ex3data1.mat`, where each training example is a 20 pixel by 20 pixel grayscale image of the digit. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. The 20 by 20 grid of pixels is "unrolled" into a 400-dimensional vector. Each of these training examples becomes a single row in our data matrix

X. This gives us a 5000 by 400 matrix X where every row is a training example for a handwritten digit image.

$$X = \begin{bmatrix} - (x^{(1)})^T - \\ - (x^{(2)})^T - \\ \vdots \\ - (x^{(m)})^T - \end{bmatrix}$$

The second part of the training set is a 5000-dimensional vector y that contains labels for the training set. We start the exercise by first loading the dataset. Execute the cell below, you do not need to write any code here.

```
In [2]: # 20x20 Input Images of Digits
        input_layer_size  = 400

        # 10 labels, from 1 to 10 (note that we have mapped "0" to label 10)
        num_labels = 10

        #  training data stored in arrays X, y
        data = loadmat(os.path.join('Data', 'ex3data1.mat'))
        X, y = data['X'], data['y'].ravel()

        # set the zero digit to 0, rather than its mapped 10 in this dataset
        # This is an artifact due to the fact that this dataset was used in
        # MATLAB where there is no index 0
        y[y == 10] = 0

        m = y.size
```

2.3.2 1.2 Visualizing the data

You will begin by visualizing a subset of the training set. In the following cell, the code randomly selects selects 100 rows from X and passes those rows to the displayData function. This function maps each row to a 20 pixel by 20 pixel grayscale image and displays the images together. We have provided the displayData function in the file utils.py. You are encouraged to examine the code to see how it works. Run the following cell to visualize the data.

```
In [3]: # Randomly select 100 data points to display
        rand_indices = np.random.choice(m, 100, replace=False)
        sel = X[rand_indices, :]

        utils.displayData(sel)
```

2.3.3 1.3 Vectorizing Logistic Regression

You will be using multiple one-vs-all logistic regression models to build a multi-class classifier. Since there are 10 classes, you will need to train 10 separate logistic regression classifiers. To make this training efficient, it is important to ensure that your code is well vectorized. In this section, you will implement a vectorized version of logistic regression that does not employ any `for` loops. You can use your code in the previous exercise as a starting point for this exercise.

To test your vectorized logistic regression, we will use custom data as defined in the following cell.

```
In [4]: # test values for the parameters theta
        theta_t = np.array([-2, -1, 1, 2], dtype=float)

        # test values for the inputs
```