Programming Exercise 3: Multi-class Classification and Neural Networks

Thiago Raulino Dal Pont
Automation and Systems Department
Federal University of Santa Catarina

May 30, 2022

Abstract

1 Introduction

In this third programming exercise, we are going to focus on Multiclass classification using Logistic regression (LR) and regularization, and start to work with Neural Networks (NNs).

Based on the Python Notebook available in Moodle, we implement some core functions to train multi-class LR and make predictions using a pre-trained NN.

This work is divided as follows:

- Section 2 presents the required materials, methods and studies to finish the exercise;
- Section 3 presents the implementations, results and discussions.

2 Methods and Materials

Before diving in the code, the exercise required some previous study in the Python libraries: `numpy`, *texttt* and `matplotlib`. To fulfill this goal, the documentations [1, 2, 3] and some Youtube videos [4, 5] were used. A study on the Linear Regression using the reference book [6] was also required.

For model evaluation, we applied accuracy to estimate how well the models fit the train data:

$$Acc = 100 \cdot \frac{|y_{act} - y_{pred}|}{y_{act}} \tag{1}$$

In terms of tools versions, we used:

- Python 3.9.7
- Numpy 1.20.3
- Matplotlib 3.5.1
- Scipy 1.7.3

3 Experiments

3.1 Multi-class Classification

In this first experiment, we implement the Logistic Regression and apply it on the digit recognition using the MNIST dataset.

The dataset is composed of images of 20x20 pixels And, to better understand the data, we present it on Figure 1.

After visualizing the data, we re-implement the cost function (with regularization) and the gradient using only linear algebra operations over matrices.

Then, we implement a one-vs-all classifier using several LR classifiers, one for each label, i.e., digit.

Figure 1: Visualizing a sample from the MNIST Dataset

Listing 1: Training One vs All classifier

```
def oneVsAll(X, y, num_labels, lambda_):

    # Some useful variables
    m, n = X.shape

    # You need to return the following variables correctly
    all_theta = np.zeros((num_labels, n + 1))

    # Add ones to the X data matrix
    X = np.concatenate([np.ones((m, 1)), X], axis=1)

    print("Training Multi-class Logistic Regression")
    for k in range(num_labels):
        print("=", end="")
        # Set Initial theta
        initial_theta = (np.random.random(n + 1)-0.5)/10

        # Set options for minimize
        options = {
            'maxiter': 100
        }

        # Run minimize to obtain the optimal theta. This function will
        # return a class object where theta is in `res.x` and cost in `res.fun`
        res = optimize.minimize(lrCostFunction,
                                initial_theta,
                                (X, (y == k), lambda_),
                                jac=True,
                                method='TNC',
                                options=options)
        theta_k = res.x
        all_theta[k,:] = theta_k


    return all_theta
```

Finally, we implement the prediction function for the one vs all classifier, using only linear algebra operations over matrices.

Listing 2: Making predictions using One vs All classifier

```python
def predictOneVsAll(all_theta, X):

    m = X.shape[0];
    num_labels = all_theta.shape[0]

    # You need to return the following variables correctly
    p = np.zeros(m)

    # Add ones to the X data matrix
    X = np.concatenate([np.ones((m, 1)), X], axis=1)

    # Calculing probabilities for h_theta for all examples in X
    h_theta = X @ all_theta.T

    # Get the categorical prediction using the argmax.
    p = np.argmax(h_theta, axis=1)

    return p
```

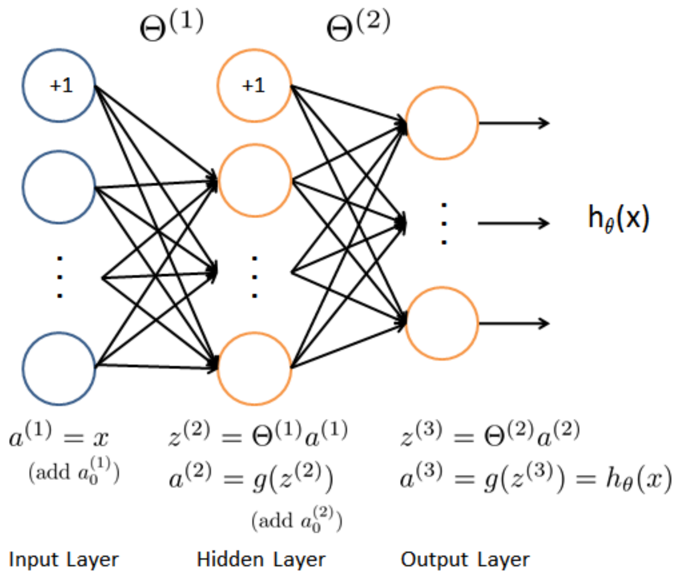
Using the test set, we achieve an accuracy of 95.84%. This is a good accuracy for a simple technique as LR is.

3.2 Neural Networks

In this section, we use the same dataset from the previous experiment, focusing on applying a pre-trained weights for a NN. Our task is to implement the function that make predictions on those weights.

The NN architecture is presented in Figure 2, where the input layer as 400 neurons, the hidden layer has 20 and the output layer has 10 neurons with sigmoid activation function.

Figure 2: NN architecture



The prediction function is presented in the listing:

Listing 3: Making predictions using Neural Network

```python
def predict(Theta1, Theta2, X):

    # Make sure the input has two dimensions
    if X.ndim == 1:
        X = X[None]  # promote to 2-dimensions

    # useful variables
    m = X.shape[0]
    num_labels = Theta2.shape[0]

    # You need to return the following variables correctly
    p = np.zeros(X.shape[0])
```

```python

    # ===================== YOUR CODE HERE =====================
    # Add intercept term to X
    X = np.concatenate([np.ones((m, 1)), X], axis=1)

    # Calculate NET z2
    z2 = Theta1 @ X.T

    # Calculate activation using sigmoid
    a2 = utils.sigmoid(z2)

    # Add bias term to a2
    a2 = np.concatenate([np.ones((1, m)), a2], axis=0)

    # Calculate NET z3
    z3 = Theta2 @ a2

    # Calculate activation using sigmoid
    a3 = utils.sigmoid(z3)

    p = np.argmax(a3, axis=0)
    # ============================================================
    return p
```

Using the pre-trained weights and the prediction function, we achieved an accuracy of 97.5% in the test set, i.e., 2% higher than loggistic regression.

References

[1] "Mathematical functions - numpy." https://numpy.org/doc/stable/reference/routines.math.html. Accessed 30 Apr 2022.

[2] "Matplotlib 3.5.1 documentation." https://matplotlib.org/3.5.1/index.html. Accessed 30 Apr 2022.

[3] "Scipy api - scipy v1.8.0." https://docs.scipy.org/doc/scipy/reference/. Accessed 12 May 2022.

[4] "Complete python numpy tutorial (creating arrays, indexing, math, statistics, reshaping)." https://www.youtube.com/watch?v=GB9ByFAIAH4. Accessed 30 Apr 2022.

[5] "Intro to data visualization in python with matplotlib! (line graph, bar chart, title, labels, size)." https://www.youtube.com/watch?v=DAQNHzOcO5A. Accessed 30 Apr 2022.

[6] S. Haykin, *Neural networks and Learning Machines*. 2008.