# Machine Learning Programming Exercise 3: Multi-class Classification and Neural Networks

adapted to Python language from Coursera/Andrew Ng

25 septembre 2020

## 1 Introduction

In this exercise, you will implement one-vs-all logistic regression and neural networks to recognize hand-written digits. Before starting the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics. To get started with the exercise, download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the cd command in Python/Spyder to change to this directory before starting this exercise.

## 2 Files included in this exercise

— **ex3.py** - Python script that will help step you through the exercise
— **ex3_nn.py** - Python script for the later parts of the exercise
— **ex3data1.mat** - Dataset for linear regression with one variable
— **ex3weights.mat** - Initial weights for the neural network exercise
— **displayData.py** - Function to help visualize the dataset
— **sigmoid.py** - Sigmoid function
⋆ **lrCostFunction.py** - Logistic regression cost function
⋆ **lrCostGradient.py** - Logistic regression cost gradient
⋆ **learnOneVsAll.py** - Train a one-vs-all multi-class classifier
⋆ **predictOneVsAll.py** - Predict using a one-vs-all multi-class classifier
⋆ **predict.py** - Neural network prediction function

[⋆] indicates files you will need to complete

Throughout the exercise, you will be using the scripts **ex3.py** and **ex3_nn.py**. These scripts set up the dataset for the problems and make calls to functions that you will write. You do not need to modify these scripts. You are only required to modify functions in other files, by following the instructions in this assignment.

## 3 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've learned can be used for this classification task. In the first part of the exercise, you will extend your previous implemention of logistic regression and apply it to one-vs-all classification.

### 3.1 Dataset

You are given a data set in ex3data1.mat that contains 5000 training examples of handwritten digits. The .mat format means that the data has been saved in a native Octave/Matlab matrix format, instead of a text (ASCII) format like a csv-file. These matrices can be read directly into your program by using the **scipy.io.loadmat** command :

```python
# Load saved matrices from file
datafile = 'ex3data1.mat'
mat = scipy.io.loadmat( datafile )
X, y = mat['X'], mat['y']
# X is an array (5000 x 400)
# y is an array (5000 x 1)
```

The matrices $X$ and $y$ will now be in your Python environment. 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 become a single row in our data matrix $X$. This gives us a $m = 5000$ by $n = 400$ matrix $X$ where every row is a training example for a handwritten digit image.

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

where $x^{(i)}_{i=1,\ldots,m}$ is a $n$-dimensional vector. The second part of the training set is a 5000-dimensional vector $y$ that contains labels for the training set. We have mapped the digit zero to the value ten. Therefore, a « 0 » digit is labeled as 10, while the digits « 1 »to « 9 »are labeled as « 1 » to « 9 »in their natural order.

## 3.2 Visualizing the data

You will begin by visualizing a subset of the training set. In Part 1 of **ex3.py**, 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, and you are encouraged to examine the code to see how it works. After you run this step, you should see an image like Figure 1.

## 3.3 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 last exercise as a starting point for this exercise.

### 3.3.1 Vectorizing the cost function

We will begin by writing a vectorized version of the cost function. Recall that in (unregularized) logistic regression, the cost function is :

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right].$$

To compute each element in the summation, we have to compute $h_\theta\left(x^{(i)}\right)$ for every example $i$, where $h_\theta\left(x^{(i)}\right) = g\left(\theta^T x^{(i)}\right)$ and $g(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. It turns out that we can compute this quickly for all our examples by using matrix multiplication. Let us define $X$ and $\theta$ as

$$X = \begin{bmatrix} — \left(x^{(1)}\right)^T — \\ — \left(x^{(2)}\right)^T — \\ \vdots \\ — \left(x^{(m)}\right)^T — \end{bmatrix} \text{ and } \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Then, by computing the matrix product $X\theta$, we have

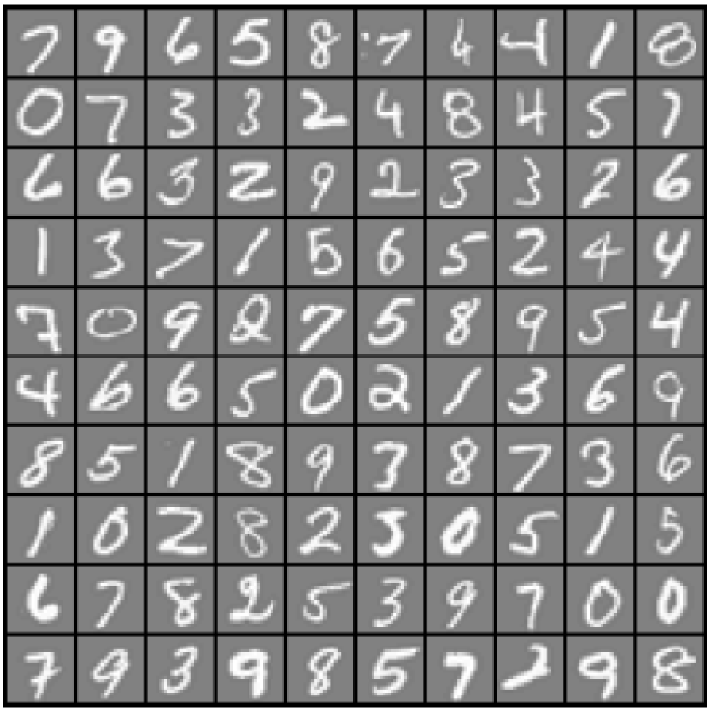

Figure 1: Examples from the dataset

FIGURE 1 – Examples from the dataset.

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

In the last equality, we used the fact that $a^Tb = b^Ta$ if $a$ and $b$ are vectors. This allows us to compute the products $\theta^T x^{(i)}$ for all our examples $i$ in one line of code. Your job is to write the unregularized cost function in the file **lrCostFunction.py**. Your implementation should use the strategy we presented above to calculate $\theta^T x^{(i)}$. You should also use a vectorized approach for the rest of the cost function. A fully vectorized version of **lrCostFunction.py** should not contain any loops. Hint : You might want to use the element-wise multiplication operation (*) and the sum operation sum when writing this function). The matrix multiplication operation is either @ or numpy.dot.

#### 3.3.2 Vectorizing the gradient

Recall that the gradient of the (unregularized) logistic regression cost is a vector where the $j^{th}$ element is defined as

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

To vectorize this operation over the dataset, we start by writing out all the partial derivatives explicitly for all $\theta_j$ ,

$$
\begin{bmatrix}
\frac{\partial J(\theta)}{\partial \theta_0} \\
\frac{\partial J(\theta)}{\partial \theta_1} \\
\frac{\partial J(\theta)}{\partial \theta_2} \\
\vdots \\
\frac{\partial J(\theta)}{\partial \theta_n}
\end{bmatrix}
= \frac{1}{m}
\begin{bmatrix}
\sum_{i=1}^m \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)} \\
\sum_{i=1}^m \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_1^{(i)} \\
\sum_{i=1}^m \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_2^{(i)} \\
\vdots \\
\sum_{i=1}^m \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_n^{(i)}
\end{bmatrix}
$$

$$
= \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}
$$

$$
= \frac{1}{m} X^T \left(h_\theta\left(x\right) - y\right) \tag{1}
$$

where

$$
h_\theta\left(x\right) - y =
\begin{bmatrix}
h_\theta\left(x^{(1)}\right) - y^{(1)} \\
h_\theta\left(x^{(2)}\right) - y^{(2)} \\
\vdots \\
h_\theta\left(x^{(m)}\right) - y^{(m)}
\end{bmatrix}
$$

Note that $x^{(i)}$ is a vector, while $\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$ is a scalar (single number). To understand the last step of the derivation, let
$\beta_i = \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$ and observe that :

$$
\sum_i \beta_i x^{(i)} =
\begin{bmatrix}
| & | & & | \\
x^{(1)} & x^{(2)} & \ldots & x^{(m)} \\
| & | & | & |
\end{bmatrix}
\begin{bmatrix}
\beta_1 \\
\beta_2 \\
\vdots \\
\beta_m
\end{bmatrix}
= X^T \beta
$$

The expression above allows us to compute all the partial derivatives without any loops. If you are comfortable with linear algebra, we encourage you to work through the matrix multiplications above to convince yourself that the vectorized version does the same computations. You should now implement Equation 1 to compute the correct vectorized gradient. Once you are done, complete the function **lrCostGradient.py** by implementing the gradient.

Debugging Tip : Vectorizing code can sometimes be tricky. One common strategy for debugging is to print out the sizes of the matrices you are working with using the size function. For example, given a data matrix $X$ of size 100x20 (100 examples, 20 features) and, a vector with dimensions 201, you can observe that $X\theta$ is a valid multiplication operation, while $\theta X$ is not. Furthermore, if you have a non-vectorized version of your code, you can compare the output of your vectorized code and non-vectorized code to make sure that they produce the same outputs.

#### 3.3.3 Vectorizing regularized logistic regression

After you have implemented vectorization for logistic regression, you will now add regularization to the cost function. Recall that for regularized logistic regression, the cost function is defined as

$$
J\left(\theta\right) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2.
$$

Note that you should not be regularizing $\theta_0$ which is used for the bias term. Correspondingly, the partial derivative of regularized logistic regression cost for $\theta_j$ is defined as

$$
\frac{\partial J\left(\theta\right)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} \qquad \text{pour } j = 0
$$

$$
\frac{\partial J\left(\theta\right)}{\partial \theta_j} = \frac{1}{m} \left(\sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} + \lambda \theta_j\right) \qquad \text{pour } j \geq 1
$$

Now modify your code in **lrCostFunction** and in **lrCostGradient** to account for regularization. Once again, you should not put any loops into your code.

Python Tip : When implementing the vectorization for regularized logistic regression, you might often want to only sum and update certain elements of $\theta$. In Python, you can index into the matrices to access and update only certain elements. For example, A[ :, 3 :5] = B[ :, 1 :3] will replaces the columns 4 to 6 of A with the columns 2 to 3 from B.

### 3.4 Learning a One-vs-all classifier

In this part of the exercise, you will discover the way to make an one-vs-all classification by training multiple regularized logistic regression classifiers, one for each of the $K$ classes in our dataset (Figure 1). In the handwritten digits dataset, $K = 10$, but your code should work for any value of $K$.

You should now read carefuly the code in **learnOneVsAll.py** to train one classifier for each class. In particular, note the code return all the classifier parameters in a matrix $\Theta \in \mathbb{R}^{K\text{x}(n+1)}$, where each row of $\Theta$ corresponds to the learned logistic regression parameters for one class. This is achieved with a ≪ for ≫-loop from 1 to $K$, training each classifier independently.

Note that the $y$ argument to this function is a vector of labels from 1 to 10, where we have mapped the digit ≪ 0 ≫to the label 10 (to avoid confusions with indexing).

When training the classifier for class $k \in \{1, \ldots, K\}$, you will want a $m$-dimensional vector of labels $y$, where $y_j \in \{0, 1\}$ indicates whether the $j$-th training instance belongs to class $k$ ($y_j = 1$), or if it belongs to a different class ($y_j = 0$). You may note that logical operators is helpful for this task.

Python Tip : Logical arrays in Python are arrays which contain binary (true or false) elements. In Python, evaluating the expression $a == b$ for a vector $a$ (of size $m$x1) and scalar $b$ will return a vector of the same size as $a$ with ones at positions where the elements of $a$ are equal to $b$ and zeroes where they are different. To see how this works for yourself, try the following code in Python :

```
a = 1:10; % Create a and b
b = 3;
p = (a == b) % You should try different values of b here
print(p)
```

Furthermore, the code use **scipy.optimize.fmin_cg** for this exercise (instead of **fmin_tnc**). **fmin_cg** works similarly to **fmin_tnc**, but is more more efficient for dealing with a large number of parameters.

After you have carefully examined the code for **learnOneVsAll.py**, the script **ex3.py** will continue to use your learnOneVsAll function to train a multiclass classifier.

#### 3.4.1 One-vs-all prediction

After training your one-vs-all classifier, you can now use it to predict the digit contained in a given image. For each input, you should compute the ≪ probability ≫that it belongs to each class using the trained logistic regression classifiers. Your one-vs-all prediction function will pick the class for which the corresponding logistic regression classifier outputs the highest probability and return the class label (1, 2,..., or $K$) as the prediction for the input example.

You should now complete the code in **predictOneVsAll.py** to use the one-vs-all classifier to make predictions.

Once you are done, **ex3.py** will call your predictOneVsAll function using the learned value of $\Theta$. You should see that the training set accuracy is about 96.5% (i.e., it classifies 96.5% of the examples in the training set correctly).

## 4 Neural networks

In the previous part of this exercise, you implemented multi-class logistic regression to recognize handwritten digits. However, logistic regression cannot form more complex hypotheses as it is only a linear classifier. [1]

In this part of the exercise, you will implement a neural network to recognize handwritten digits using the same training set as before. The neural network will be able to represent complex models that form non-linear hypotheses. For this week, you will be using parameters from a neural network that we have

1. You could add more features (such as polynomial features) to logistic regression, but that can be very expensive to train.

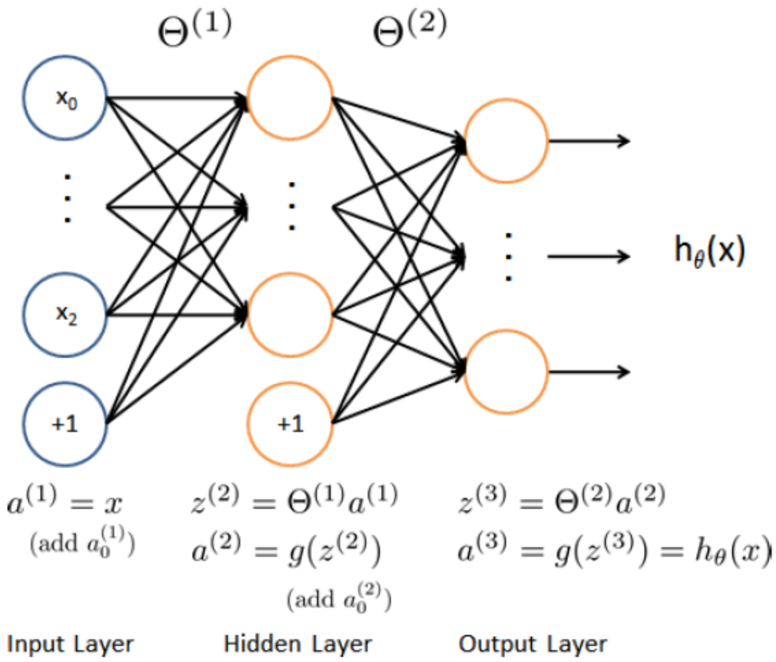


FIGURE 2 – Neural network model.

already trained. Your goal is to implement the feedforward propagation algorithm to use our weights for prediction. In next week's exercise, you will write the backpropagation algorithm for learning the neural network parameters. The provided script, `ex3\_nn.py`, will help you step through this exercise.

### 4.1 Model representation

Our neural network is shown in Figure 2. It has 3 layers : an input layer, a hidden layer and an output layer. Recall that our inputs are pixel values of digit images. Since the images are of size 20x20, this gives us 400 input layer units (excluding the extra bias unit which always outputs +1). As before, the training data will be loaded into the variables $X$ and $y$.

You have been provided with a set of network parameters $(\Theta^{(1)}; \Theta^{(2)})$ already trained by us. These are stored in ex3weights.mat and will be loaded by **ex3_nn.py** into `Theta1` and `Theta2`. The parameters have dimensions that are sized for a neural network with 25 units in the second layer and 10 output units (corresponding to the 10 digit classes).

```
# Load the weights into variables Theta1 and Theta2
data = scipy.io.loadmat('ex3weights.mat')
Theta1 = data['Theta1'] % Theta1 has size 25 x 401
Theta2 = data['Theta2'] % Theta2 has size 10 x 26
```

### 4.2 Feedforward Propagation and Prediction

Now you will implement feedforward propagation for the neural network. You will need to complete the code in **predictNeuralNetwork.py** to return the neural network's prediction. You should implement the feedforward computation that computes $h_\theta\left((x^{(i)})\right)$ for every example $i$ and returns the associated predictions. Similar to the one-vs-all classification strategy, the prediction from the neural network will be the label that has the largest output $h_\theta\left((x)\right)_k$.

Implementation Note : The matrix $X$ contains the examples in rows. When you complete the code in **predictNeuralNetwork.py**, you will need to add the column of 1's to the matrix. The matrices Theta1 and Theta2 contain the parameters for each unit in rows. Specifically, the first row of Theta1 corresponds to the first hidden unit in the second layer.In Python, when you compute $z^{(2)} = \Theta^{(1)}a^{(1)}$, be sure that you index (and if necessary, transpose) $X$ correctly so that you get $a^{(l)}$ as a column vector.

Once you are done, **ex3_nn.py** will call your **predicNeuralNetwork** using the loaded set of parameters for `Theta1` and `Theta2`, and $\lambda = 1$. You should see that the accuracy is about 97.5%. After that, an interactive sequence will launch displaying images from the training set one at a time, while the console prints out the predicted label for the displayed image. To stop the image sequence, press Ctrl-C.

## 4.3 Some questions, you have to answer...

Le codage n'est qu'un prétexte pour comprendre les différents concepts du machine learning. **Ici aucune équation...Exprimez avec vos mots les concepts pour les comprendre.**

— Dans ce TP, l'approche multiclasse est choisie mais pour quelle raison utilise-t-on cette approche ?
— Quelles sont les différentes possibilités pour cette approche ?
— Décrivez brièvement l'approche basée sur les réseaux de neurones.

## 4.4 Conclusion

Pensez à m'envoyer ou déposer sur Moodle une archive avec vos codes nommée avec votre (vos) nom(s).