UNIVERSIDADE DE AVEIRO
DEPARTAMENTO DE ELECTRÓNICA TELECOMUNICAÇÕES E INFORMÀTICA

Machine Learning (2018/19) – Lab work 4

Objectives: Multiclass classification. Implementation of one-vs-all logistic regression (linear classification) and neural networks (nonlinear classification) to recognize hand-written digits (from 0 to 9) from images.

First, you need to download the starter code to the directory where you wish to complete the exercise.

Part 1 One-vs-all logistic regression (linear classification)

Files included in part 1
ex3.m - Octave/MATLAB script that steps you through part 1 (the main program for part 1)
ex3data1.mat - Training set of hand-written digits
displayData.m - Function to help visualize the dataset
fmincg.m - Function minimization routine (similar to fminunc)
costFunctionReg.m – function you completed for lab work 3
oneVsAll.m - Train a one-vs-all multi-class classifier **(you need to finish this function)**
predictOneVsAll.m - Predict using a one-vs-all multi-class classifier **(you need to finish this function)**

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

1.1 Data set

ex3data1.mat contains 5000 training examples of handwritten digits. This is a subset of the MNIST handwritten digit dataset (http://yann.lecun.com/exdb/mnist/). 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. After loading this file, you will get directly the training matrices *X* and *y*.
There are 5000 training examples, 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 the data matrix *X*. This gives a 5000 by 400 matrix *X* where every row is a training example for a handwritten digit image.

The second part of the training set is a 5000-dimensional vector *y* that contains labels for the training set. To make things more compatible with Octave/MATLAB indexing, where there is no zero index, we have mapped the digit 0 to the value 10. Therefore, a 0 digit is labeled as 10, while the digits 1 to 9 are labeled as 1 to 9 in their natural order.

1.2 Visualizing the data

You will begin by visualizing a subset of the training set. The code randomly 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, you should see an image like Fig. 1.

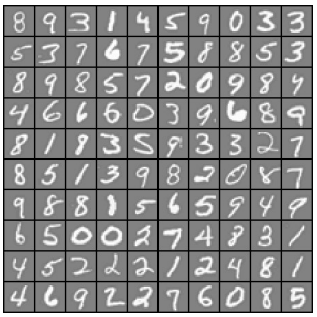

Fig.1 Examples from the dataset

1.3 One-vs-all regularized logistic regression classifiers

You will implement one-vs-all classification by training multiple regularized logistic regression classifiers, one for each of the K classes in our dataset.

In the handwritten digits dataset, K = 10, but your code should work for any value of K. You should complete the code in *oneVsAll.m* to train one classier for each class. In particular, your code should return all the classifier parameters in a Kx(N+1) matrix θ, where each row of θ corresponds to the learned logistic regression parameters for one class. You can do this with a for-loop from 1 to K, training each classier 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 classier for class $k \in \{1,2,...K\}$ you will want m-dimensional vector of labels *y*, where $y_j \in \{0,1\}$ indicates whether the j^{th} training instance belongs to class *c* ($y_j = 1$) or if it belongs to a different class ($y_j = 0$). Logical arrays are helpful for this task.

Octave/MATLAB Tip: Logical arrays in Octave/MATLAB are arrays which contain binary (0 or 1) elements. In Octave/MATLAB, evaluating the expression *v == b* for a vector *v* and scalar *b* will return a vector of the same size as *v* with 1 at positions where the elements of *v* are equal to *b* and 0 where they are different. To see how this works for yourself, try the following code in Octave/MATLAB:

```
v = 1:10; % Create a and b
b = 3;
v== b % You should try different values of b
```

For this exercise we use *fmincg* (instead of *fminunc*) to optimize the parameters θ. *fmincg* (optimization with conjugate gradient algorithm) works similarly to *fminunc*, but is more efficient for dealing with a large number of parameters. Call *fmincg* with function *costFunctionReg.m*, the way it was done in lab work 3.

1.4 One-vs-all Prediction

After training the one-vs-all classier, 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 assigned class for this example.

Complete the code in *predictOneVsAll.m* to use the trained (one-vs-all) classifiers to make predictions. You should see that the training set accuracy is about 94.9% (i.e., it classifies 94.9% of the examples in the training set correctly).

Part 2. One-vs-all with neural network classifier (nonlinear classification)

Files included in part 2
ex3 nn.m - Octave/MATLAB script guides you through part 2 ((the main program for part2)
ex3data1.mat - Training set of hand-written digits
ex3weights.mat - Initial weights for the neural network exercise
displayData.m - Function to help visualize the dataset
predict.m - Neural network prediction function **(you need to finish this function)**

In the previous part, you implemented multi-class logistic regression to recognize handwritten digits. However, logistic regression cannot propose more complex hypotheses as it is only a linear classifier. In this part, you will implement a neural network to recognize handwritten digits using the same training set as before. The neural network is able to represent complex non-linear hypotheses (models). For this exercise, you will use parameters from a neural network that have been already trained. Your goal is to implement the feedforward propagation algorithm to use the learned parameters (weights) for prediction.

2.1 Model representation

The neural network is shown in Fig. 2. It has 3 layers (an input layer, a hidden layer and an output layer). Recall that the inputs are pixel values of digit images. Since the images are of size 20x20, this gives 400 input layer units (excluding the extra bias unit which always outputs +1). As before, the training data will be directly loaded into the variables X and y. In file *ex3weights.mat* are stored the trained network parameters and will be loaded as *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). *Theta1* has size 25x401 and *Theta2* has size 10x26.

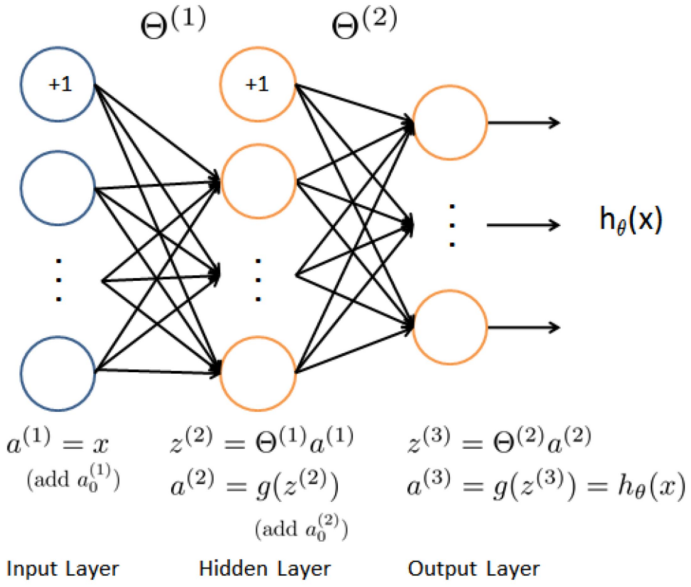

Fig. 2 Neural Network model

2.2 Feedforward Propagation and Prediction

Now you will implement feedforward propagation for the neural network. Complete the code in *predict.m* to return the neural network's prediction. You should implement the feedforward computation that computes the NN output for every example 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.

Implementation Note: The matrix X contains the examples in rows. When you complete the code in *predict.m*, 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, compute $z2=a1*(Theta1)^T$ and $z3=a2*(Theta2)^T$. Check if *a2* and *a3* are column vectors.

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.