Report Lab4 for the Machine Learning course

Nicholas Attolino

December 2023

Contents

Introduction **3**

1 Goal **3**
1.1 Data Files overview 4

2 Methods used **4**
2.1 Classify Patterns with a Neural Network 4
2.2 Autoencoders 5
2.3 The Matlab Program 5

3 Results **6**
3.1 Pattern classification 6
3.2 The trained autoencoder 7

4 Conclusions **10**

Introduction

This work is a report for the fourth laboratory of the Machine Learning course. The subject of this laboratory are the **Neural Networks**. The programs and the tests are implemented in MATLAB, a versatile platform that serves as a numerical computation and statistical analysis environment, as well as a programming language[3]. The chapters are divided in:

1. Goal;
2. Methods used;
3. Results.

1 Goal

The objective of this project is to get acquainted with the MATLAB Neural Network Toolbox trying to classify some patterns from chosen datasets and ultimately to train a multi-layer perceptron as an autoencoder for the *MNIST* dataset.
The Deep Learning Toolbox offers powerful tools like the following:

- The **Neural Network Fitting App**, a system that, given a dataset with a very simple configuration, will train a shallow network to fit the given data;
- The **Neural Network Pattern Recognition App** that can train a shallow neural network to classify patterns;
- The **Neural Network Clustering app**;
- The **Neural Network Time series app**.

In this project the firsts 2 tools were used.

1.1 Data Files overview

```
- Attolino-Lab4.zip
    - Attolino-Lab4
        - Task1
            - iris_data.txt
            - Task1.m
            - wine_data.txt
        - Task2
            - class_pair_plots
                - ...
            - mnist
                - ...
            - plotcl.m
            - Task2.m
        - Task0.m
```

In the zip archive there are the files shown above, where we find:

- Task0 MATLAB code;
- Task1 MATLAB code;
- iris_data is the first dataset for Task1;
- wine_data is the second dataset for Task1;
- Task2 MATLAB code;
- plotcl is the function to plot data;
- mnist is the dataset folder for Task2;
- class_pair_plots is a folder with some encoded data plotted.

2 Methods used

2.1 Classify Patterns with a Neural Network

Thanks to the Neural Network Pattern Recognition tool, it's possible to train a shallow network with custom datasets.
The tool can either be used through a graphical interface or by command line.
This work rely on the Iris[1] dataset and on the Wine[2] dataset: the *Iris* dataset has 150 instances, 4 numeric attributes and 3 classes; while the *Wine* dataset is a 3-class dataset with at least 48 occurrences per class and 13 continuous attributes.

2.2 Autoencoders

Training an autoencoder involves training a multi-layer perceptron neural network where the desired target aligns with the provided input.
Consequently, the network assimilates a compressed representation of the data, enabling the hidden layer to recognize and categorize passively certain patterns. The data object of this study are hand-written digits in a 28 by 28 grey-scale matrix, while the autoencoder needs 784 input and output units and the hidden layer has 2 units.

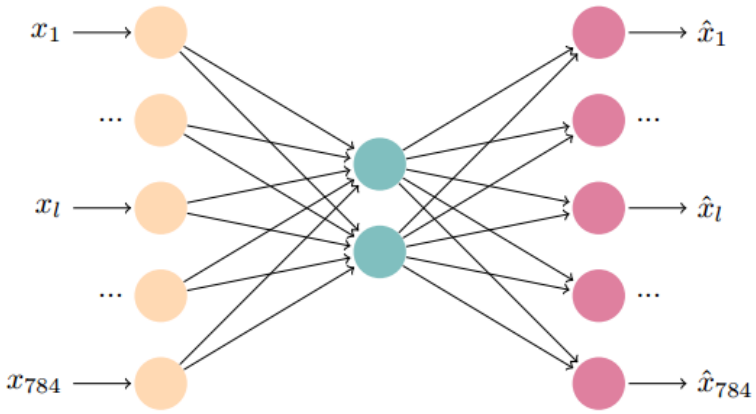

Figure 1: Autoencoder Neural Network

2.3 The Matlab Program

The developed program conducts training for an autoencoder using a training set that consists of only 2 categories of handwritten digits. Subsequently, it encodes the same dataset through the trained autoencoder. Plotting the encoded data will show the pattern recognition ability of the network.
Both the training set and test sets undergo processing via a custom function named *data_pre_processing*. This function requires input specifying the two desired categories and the percentage of available observations to utilize. It generates a matrix containing the specified observations and arranges the matrix in a random order.
To plot the encoded data is used the *plotcl* function: this takes the encoded observations and their classes and plots them with a color that depends on the class they belong.
It's crucial to highlight that the autoencoder remains unaware of the labels assigned to the observations throughout its operation. These labels serve solely for the graphic feedback provided by the *plotcl* function since distinguishing the classes allows understanding if the autoencoder has been able to recognize any patterns.

3 Results

3.1 Pattern classification

The developed program empowers the user to select which one of them shall be used to train the network.
Following this choice, a series of minor preprocessing instructions are carried out. The mentioned tool requires two matrices: the predictors matrix contains the essential input data crucial for predictions, while the responses matrix holds the corresponding classes for each observation.
After several simple operations, both matrices will be ready. For the network to function effectively, it necessitates information on the desired training function among the available options, the dimensions of the hidden layer, as well as the sizes allocated for the training, validation, and test sets.
Once these parameters are established, the program proceeds to execute the training and testing phases.
The training process persists until a specific stopping condition is fulfilled. In these instances, the training persists until there is a consecutive rise in validation error for six iterations (*Met validation criterion*). When this happens it means the network is probably overfitting.
The tool itself offers multiple plots to understand the results of the test, like the Confusion matrices in Figure 2 and the ROC Curves in Figure 3.

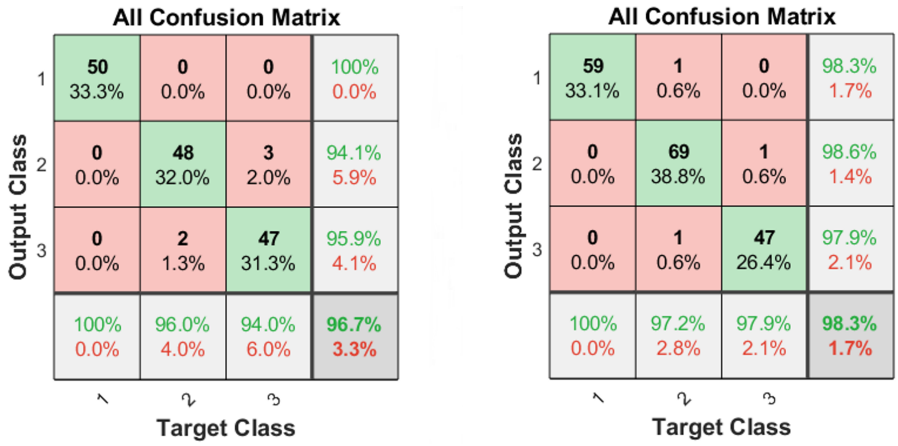

Figure 2: Confusion matrices for the Iris and the Wine datasets respectively

The accuracy of the network's outputs can be observed by examining the number of correct classifications, depicted in the green (diagonal) squares. These squares indicate instances where the output class aligns with the target class. On the other hand, the red squares highlight chases where the network incorrectly assigned observations to a different class.

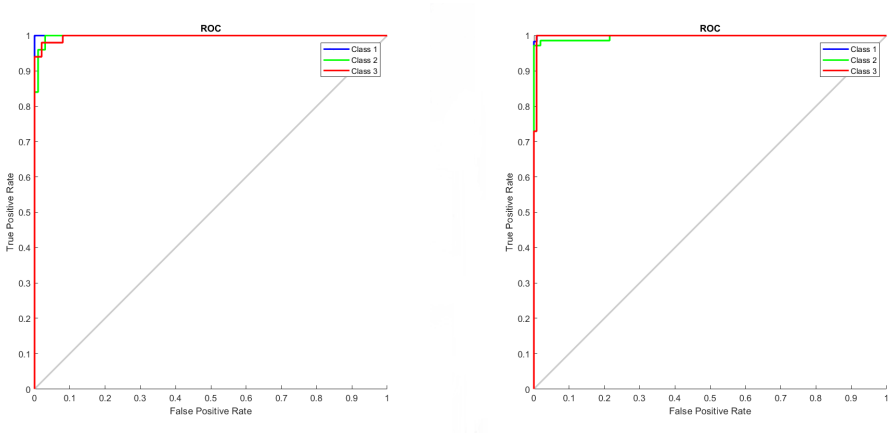

Figure 3: ROC Curves for the Iris and the Wine datasets respectively

The colored lines represent the ROC curves. The ROC curve is a plot of the true positive rate (sensitivity) versus the false positive rate (1 - specificity) as the threshold is varied.
A perfect test would show points in the upper-left corner, with 100% sensitivity and 100% specificity. For these problems, the networks performs very well.

3.2 The trained autoencoder

The test has been carried out on 7 pairs of classes. The function loops on a matrix with the 7 pairs as rows, and executes the training and the encoding.

$$Classes = \begin{bmatrix} 1 & 8 \\ 10 & 8 \\ 10 & 1 \\ 2 & 6 \\ 2 & 9 \\ 4 & 1 \\ 1 & 6 \end{bmatrix}$$

For the training, the set had about 60% of the available data, meanwhile for the test, only a 5% of data was used.

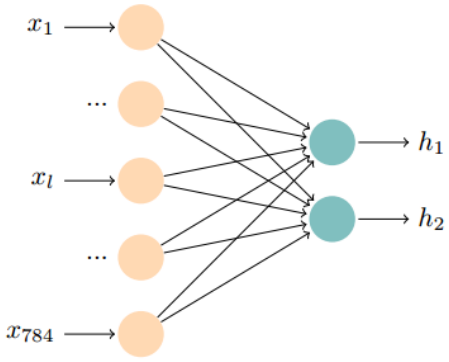

Figure 4: Encoder

By plotting the encoded data of every pair of classes, a scatter plot in two dimensions shows the encoder's capability for recognizing patterns.

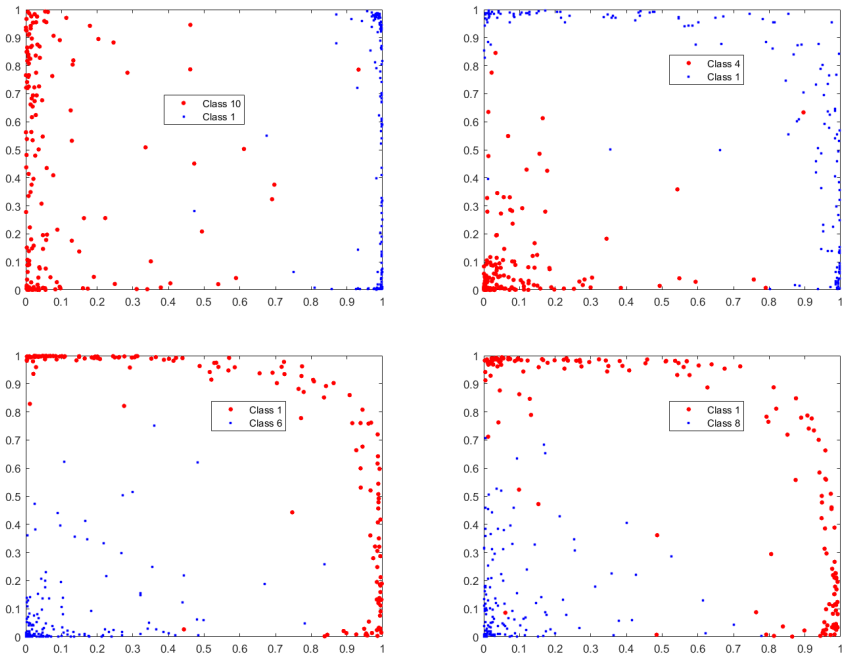

Figure 5: Plots of encoded data with some almost clear decision regions

For some digits the work is tougher and though there are not well shaped regions:

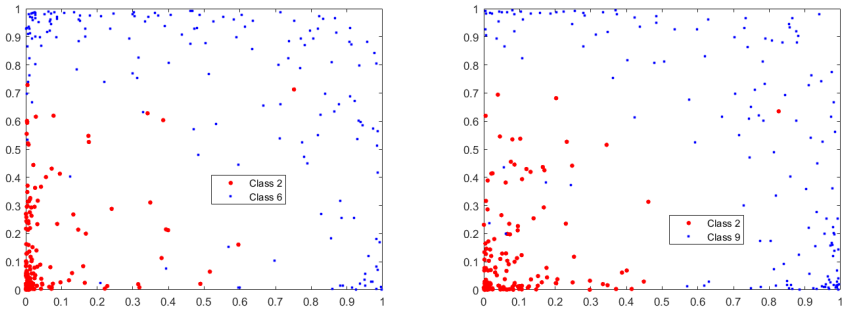

Figure 6: Plots of encoded data with blurry decision regions

It's noteworthy that these outcomes were achieved without the network having prior knowledge of the specific class to which each individual observation belongs.
This represents an instance of unsupervised learning, where the autoencoder autonomously identified certain patterns by attempting to replicate the input as the output.
This occurred subsequent to a compression phase caused by a hidden layer containing fewer units than the input/output layer.

4 Conclusions

The MATLAB Deep Learning Toolbox demonstrated its immense utility in this context.
It enables swift and straightforward training of neural networks, serving both fitting and pattern recognition objectives (those are the object of this work). Moreover, the provided visualizations aid in assessing a trained neural network. The autoencoder is a concrete proof of how the neural networks can distinguish different classes of data without a prior clue about their differences.
A possible next step involves using stacked autoencoders[4] in order to have a network with multiple hidden layer trained one by one.

References

[1] R. A. Fisher, *Iris*. 1988, DOI: https://doi.org/10.24432/C56C76.

[2] S. Aeberhard and M. Forina, *Wine*. 1991, DOI: https://doi.org/10.24432/C5PC7J.

[3] T. M. Inc., *MATLAB Version: 23.2.0.2365128 (R2023b)*. The MathWorks Inc., 2023.

[4] *Stacked Autoencoders*. [Online]. Available: `https://it.mathworks.com/help/deeplearning/ug/train-stacked-autoencoders-for-image-classification.html`.