NEURAL NETWORKS

Machine Learning 2018/19: Assignment 4

Torielli Davide   Fusaro Fabio
EMARO - European Master on Advanced Robotics
DIBRIS - Dip. Informatica, Bioingegneria, Robotica, Ing. Dei Sistemi
Università Degli Studi Di Genova, February 4, 2019

I. Introduction

NEURAL NETWORK is a network of several, interconnected, simple units. It is a model for learning mapping from inputs to outputs that tries to emulate the inner mechanics of brain processing.
In this work we use the *neural network toolbox* of Matlab to solve different problems with neural networks.

II. The Theory

A. The basis

The neuron of the network is a simple unit which takes an *input* (from the outside or from another neuron), makes a *decision* (based on some function) and gives an *output* (to another neurons or to outside the network).
In math terms:

$$r = \boldsymbol{x} \cdot \boldsymbol{w}$$
$$a = f(r - \theta)$$

where:

- $\boldsymbol{x}$ is the d-dimensional vector of input to the network.
- $\boldsymbol{w}$ are the corresponding *weights*.
- r is the net input on the neuron membrane.
- $f()$ is a nonlinear function.
- θ is a threshold.
- a is the output of the neuron (*activation value* or *action potential*).

Various function $f()$ can be used, for example the *Heaviside step*, the *sigmoid*, the *hyperbolic tangent* or the *Softplus*.

Learning in (artificial) neural networks occurs, in practice, with slow modifications of the weights. So, there is a procedure that gradually changes them accordingly to experience acquired in the past, so:

$$\boldsymbol{w}_{\tau+1} = \boldsymbol{w}_{\tau} + \triangle \boldsymbol{w}_{\tau}$$

B. Multiple layers

There are various topologies for a neural network. When we have multiple layers (where each layer contains various units) we speak about *multi-layer neural network*. The easiest neural network with multiple layers has:

- One *input* layer, which takes the external inputs and spreads them to another layer, the *hidden* one.
- One (or more) *hidden* layer. It is call *hidden* because it does not take any input and does not give any output from/to the outside of the network. So it is only linked to another layers.
- The *output* layer provides the data to the outside of the network.

Neural networks can be used for various problems. Here, we concentrate on *pattern recognition*, that is, *classification*. In the first task we use a shallow neural network, with different hidden layers. The more the hidden layers are, the more the results are better, but the computation time longer.
For this problem, it is important to divide the set in *training*, *validation*, *testing*.

The training is usually the bigger one (around 70%) and it is used for the train. The validation set is used during the train to not overfit the data. Overfitting means that we train ”too well” on the data given, losing in generality (i.e. the error on the testing set will be high).
Finally, the test set is the one used to give a indication of the error that our neural network does.

C. *Autoencoder*

The *autoencoder* is a type of neural network.
The simplest *autoencoder* has one input, one hidden and one output layer.
Another characteristic is that the *autoencoder* has same number of output units and input units; units in the hidden layer **less** than the input units.
The *autoencoder* learns an **internal, compressed** representation of the data. The interesting output are the values of the hidden layer units because, in theory, similar patterns will have similar representations. So, data of the same class will produce similar weights for the encoder. For example, an image which represents an handwritten 1 will similar weights to others 1, quite similar to 7 (because written 1 is similar to written 7), but very different weights from 8.
In practice, this neural network learn to **classify**.

III. Our Work

Our work consists in the use of the Neural Network Toolbox of Matlab.
The work is divided in 2+1 tasks:

- **Task 0**: Follow the Matlab tutorial on the use of the Neural Network Toolbox, in particular the fitting data tool.
- **Task 1**: Classify a couple of patterns with different design choice, using the neural pattern recognition tool.
- **Task 2**: Use an Autoencoder.

A. *Task 1*

The data chosen are taken from *UCI machine learning repository*: Wine dataset and Wifi dataset.
After download the data, we have to organize them correctly to be used by the tool. We choose to organize the samples as columns, so data matrices are $N_{instances} \times N_{attributes}$.
For the target, the tool want a $N_{instances} \times N_{class}$ matrix where a 1 in (i, j) position means that instances i belong to class j; otherwise the value is 0 (that is, One vs All problem).
Then the neural pattern recognition tool is launched. We modify the code generated by the tool to do different experiments. We change two things: the number of the hidden neurons of the network and the algorithm to train the network (i.e. *trainscg* and *trainbr*).
The *trainscg* uses the scaled conjugate gradient method to update weight and bias values.
The *trainbr* method is the Bayesian regularization process: it minimizes a combination of squared errors and weights, and then determines the correct combination to produce a network that generalizes well.
The first method uses less memory and it is faster, the second is heavier but it is better for difficult problems.
The ratio in which divide the set in *training*, *validation*, *testing* is fixed to 70%, 15%, 15%.

1) Wine Dataset: This is a dataset of the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars (so, three different class). The number of instances are divided in:
class 1 : 59 instances;
class 2 : 71 instances;
class 3 : 48 instances;
According to the description given, this is a well posed problem, good to test new classifier, but not very challenging.
The goal is to classify wine in one of the three cultivars.

2) Wireless Indoor localization Dataset: This dataset represent wifi signals strenght observed by smartphones. Each attribute is an observation from a smartphone, that start from 0 (strong signal) and goes into negative integer values. The four possible values of the target represent in which room the wifi signal is generated.

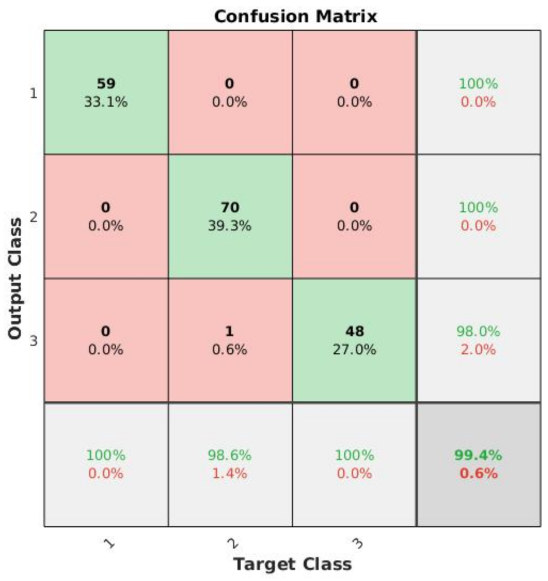
B. *Task 2*

For the autoencoder, we used the MNIST data, that are images of handwritten numbers from 0 to 9. So the targets are ten and correspond to the number written in the images (note that zero has 10 as target to not use 0 to index the matrices).
From the dataset, we use combination of only two classes (e.g. class 1 and class 7), and we train the encoder with them.
After processing correctly the data, we use *trainAutoencoder* function to train. We use an autoencoder with two units in the hidden layer. This is done because so the computations are not too long, and also because we can represent the output producing a 2D plot, easily understandable. Then, we encode the data with another function, *encode*.
At the end, we plot the output of the two hidden units: the first along x axis, the second along y axis, producing a plot with various points. The color (and shape) of these points represent the class to which the images belong. If the autoencoder learned well, images of the same class will be nearer and images of different class will be more distant, producing two good linearly separable clusters.

IV. Experimental Results

A. *Task 1*

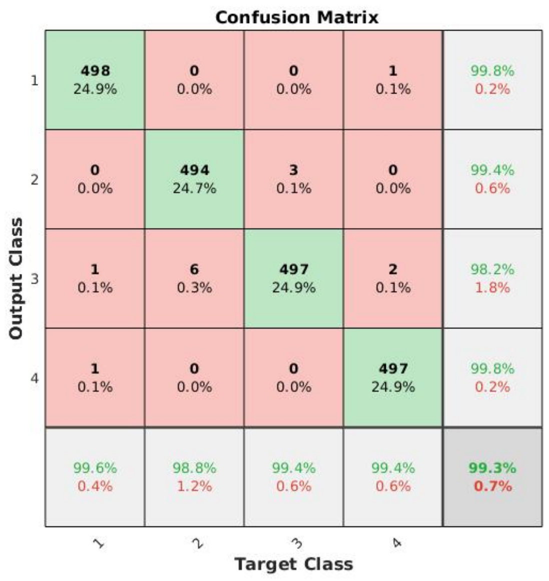
The confusion matrix represents the quality of the results.
Along the diagonal (green squares), there are the *true positive* percentages for each class. The red squares are instead the false positive/negative. The last elements in the diagonal are the accuracy (in green) and the overall error (1-accuracy). Other important parameters are the *precision* and *false discovery rate* for each predicted class. These values are in the last column and represent the percentages of all the examples predicted to belong to each class that are correctly and incorrectly classified. In the last row there are the *sensitivity* and *false negative rate* which represent the percentages of all the examples belonging to each class that are correctly and incorrectly classified.



Confusion Matrix

Output Class \ Target Class	1	2	3	
1	59 33.1%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	70 39.3%	0 0.0%	100% 0.0%
3	0 0.0%	1 0.6%	48 27.0%	98.0% 2.0%
	100% 0.0%	98.6% 1.4%	100% 0.0%	**99.4% 0.6%**

Fig. 1: *Wine set with 5 hidden units and br method*



Confusion Matrix

Output Class \ Target Class	1	2	3	4	
1	498 24.9%	0 0.0%	0 0.0%	1 0.1%	99.8% 0.2%
2	0 0.0%	494 24.7%	3 0.1%	0 0.0%	99.4% 0.6%
3	1 0.1%	6 0.3%	497 24.9%	2 0.1%	98.2% 1.8%
4	1 0.1%	0 0.0%	0 0.0%	497 24.9%	99.8% 0.2%
	99.6% 0.4%	98.8% 1.2%	99.4% 0.6%	99.4% 0.6%	**99.3% 0.7%**

Fig. 2: *Wifi set with 5 hidden units and br method*

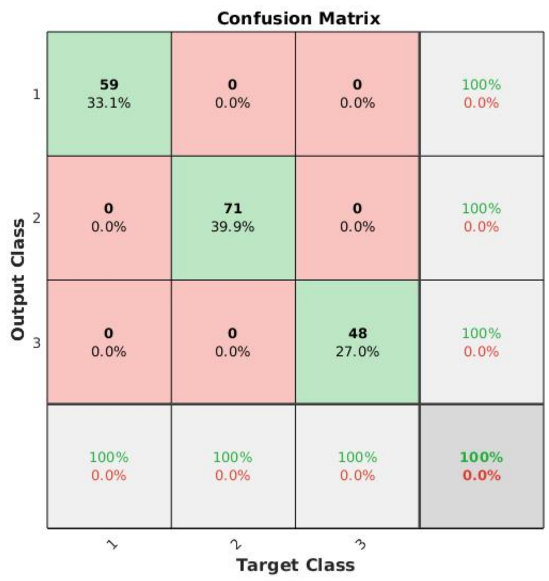


Fig. 3: *Wine set with 30 hidden units and br method*

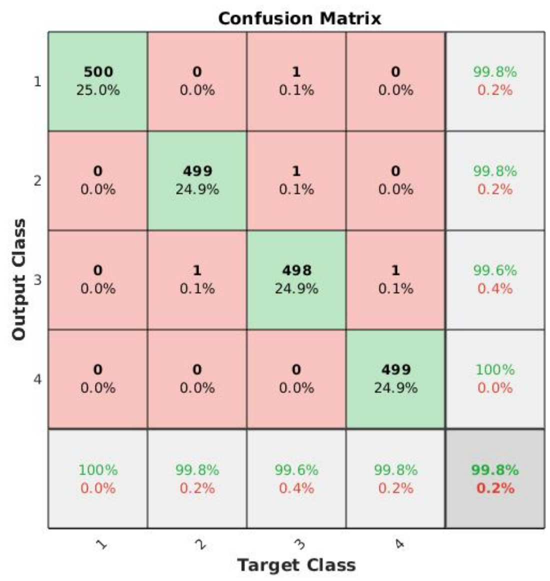


Fig. 4: *Wifi set with 30 hidden units and br method*

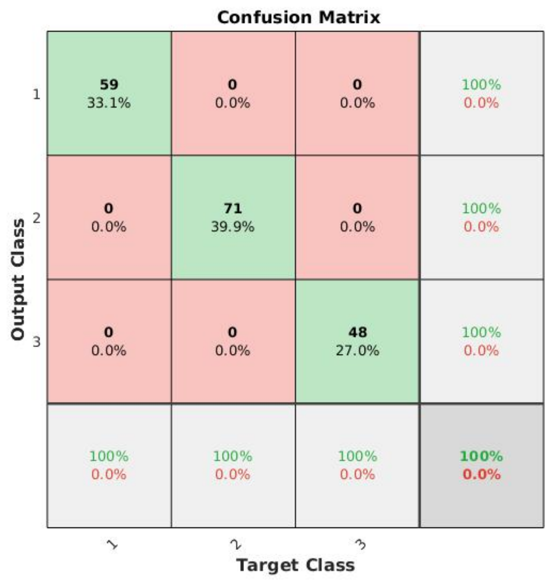


Fig. 5: *Wine set with 30 hidden units and scg method*

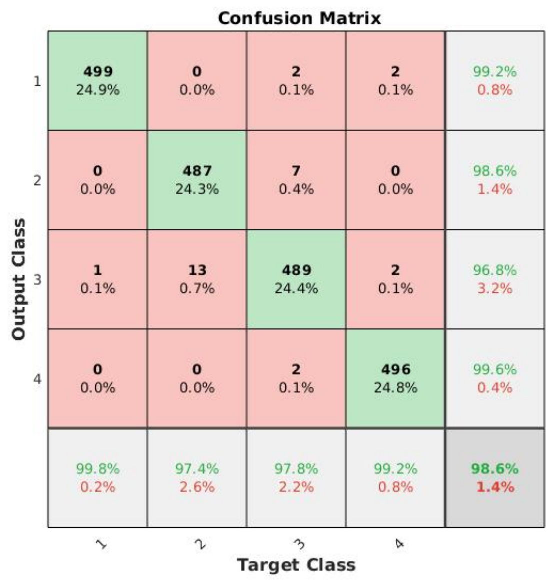


Fig. 6: *Wifi set with 30 hidden units and scg method*

In the figures above we test different algorithms with different numbers of hidden neuron units. The more the units are, the better the results are. This is in evidence in the Fig. 1 (5 hidden units) and in the Fig. 3 (30 hidden units): using the same algorithm we have an improvement of the results. The same conclusions can be done for the Fig. 2 and the Fig. 4 where only the dataset changes.
We try also to change the algorithm maintaining the same number of hidden neurons (30). As we can see in Fig. 6 with respect to the Fig. 4 for the wifi dataset, where we used respectively the *scaled conjugate gradient* and the *Bayesian regularization* algorithm, the results are less accurate. Instead for the wine dataset (Fig. 5 with respect to the Fig. 3) the results do not change. This is due to the fact that the wine dataset is an easy problem. Instead the wifi data set is a more challenging problem and so the results are better in the case of the *Bayesian regularization* algorithm (Fig. 4).

B. Task 2

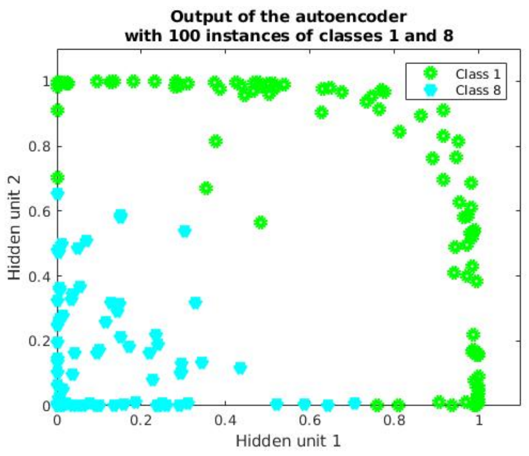


Fig. 7: *Autoencoder of classes 1 & 8*

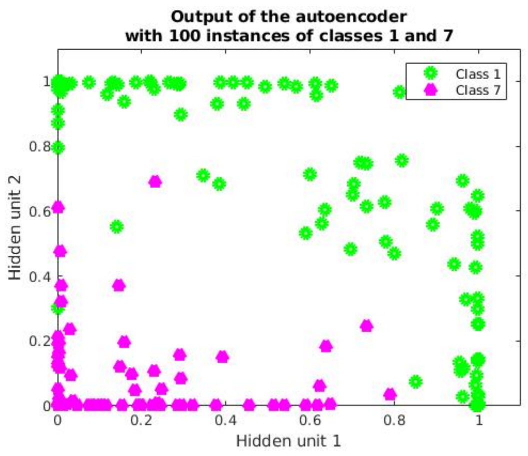


Fig. 8: *Autoencoder of classes 1 & 7*

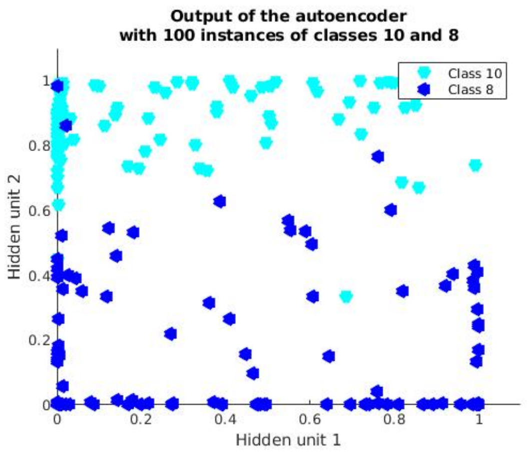


Fig. 9: *Autoencoder of classes 0 & 8*

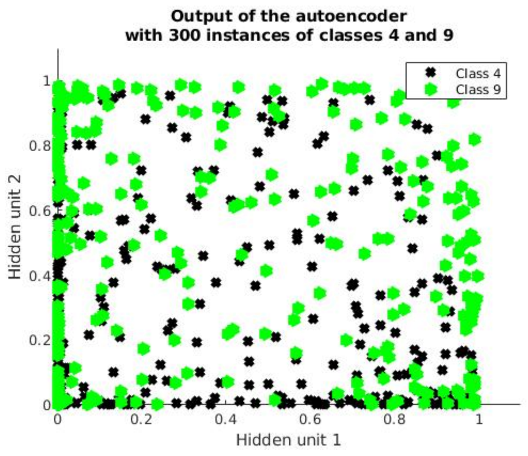


Fig. 10: *Autoencoder of classes 4 & 9*

In the task 2 results, we can see the output of the two neurons. We can appreciate that, when there are ”easy” classes to distinguish, the dots are linearly separate, e.g. the class 1 and 8 (Fig. 7). Instead, with ”difficult” pairs the results are worse (1-7 in Fig. 8 and 0-8 in Fig. 9) and even really bad for 4-9 comparison (the handwritten 4 and handwritten 9 are very similar) (Fig. 10).