NEURAL NETWORKS

Student Gianluca Galvagni, S5521188, Università degli Studi di Genova

***Abstract*—How fourth assignment, we talked about neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. Our goal for this assignment was used a Matlab's own neural network tools, contained in a library called Neural Networks Toolbox and then created the simplest auto-encoder network.**

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.
They are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.
Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.

II. 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:

$$r = x \cdot wa = f(r - \theta)$$

where:

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

Many functions **f()** can be used, for instance the *sigmoid*, the *hyperbolic tangent* or the *Heaviside*.

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:

$$w_{l+1} = w_l + \triangle w_l$$

III. Multiple layers

There are various typologies 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, I concentrate on *pattern recognition*, that is, *classification*.
In the first task we use a shallow neural network, with different hidden layers. In this case, the hidden layers are the computation time longer. For these kind problems, 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 over-fit the data. Over-fitting means that I 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.

IV. Auto-encoder

The *auto-encoder* is a type of neural network. The simplest *auto-encoder* has one input, one hidden and one output layer. Another characteristic is that the *auto-encoder* has same number of output units and input units; unit in the hidden layer **less** than the input units.
The *auto-encoder* 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**.

V. Lab activity

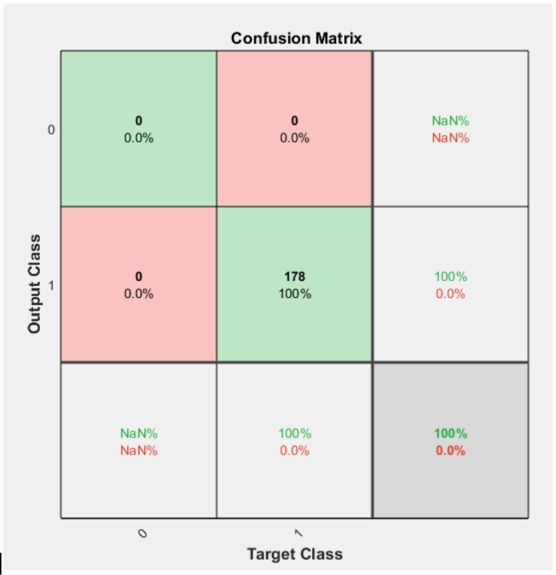
In this assignment I have used Matlab R2022b to make my program. My work consists in 3 tasks:

- **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 data sets with different design, using the neural pattern recognition tool.
- **TASK 2:** Use an Auto-encoder.

I did that:

1) *TASK 1:*
The data chosen are taken from *UCI machine learning repository*: Wine data set and Breast-Cancer-Wisconsin data set. After download the data, I had to organize them correctly to be used by the tool. While I always had patterns as rows and variables as columns, here the functions expect patterns as columns and variables as rows. So I transposed the data matrices chosen.
Then the neural pattern recognition tool is launched. I modified the code generated by the tool to do different experiments. I changed two things: the number of the hidden neurons of the network and the algorithm to train the network (i.e. *trainscg*, *trainbr* and *trainlm*). The *trainlm* and the *trainscg* are faster than the *trainbr*. The *trainscg* uses the scaled conjugate gradient method to update weight and bias values.
The *trainbr* method is he 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 ratio in which divide the set in training, validation, testing is fixed to 70%, 15%, 15%.

2) *TASK 2:*
For the auto-encoder, I used the MNIST data, that are images of handwritten numbers from 0 to 9.
So the targets were compare and correspond to the number written in the images ("1" will correspond to very similar representations, and quite similar to "7" but different from "0" or "8").
From the data-set, I used combination of only two classes (e.g. class 1 and class 7), and we train the encoder with them.
After processing correctly the data, I used *trainAutoencoder* function to train. I used an auto-encoder with two units in the hidden layer. This is done because so the computations are not too long, and also because I can represent the output producing a 2D plot, easily understandable. Then, I encoded the data with another function, *encode*.
At the end, I plotted 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 auto-encoder 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.

[a]


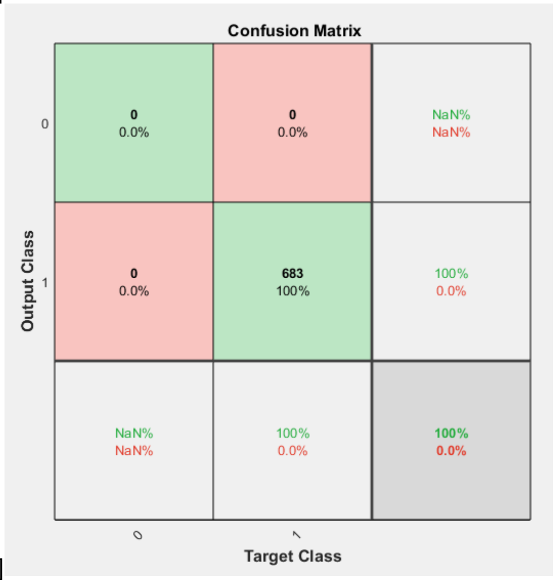
[b]


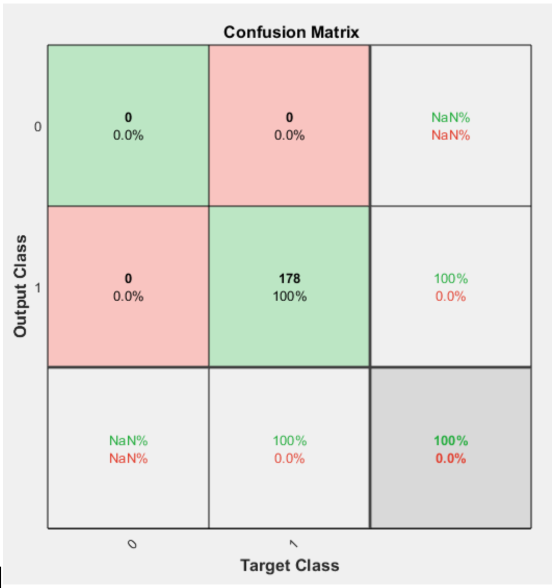
Fig. 1. CONFUSION MATRIX, represents the quality of the results.
(a) Wine set with 10 hidden units, with 5 size for each unit and *trainbr* method.
(b) BCW set with 10 hidden units, with 5 size for each unit and *trainbr* method.

VI. Conclusion

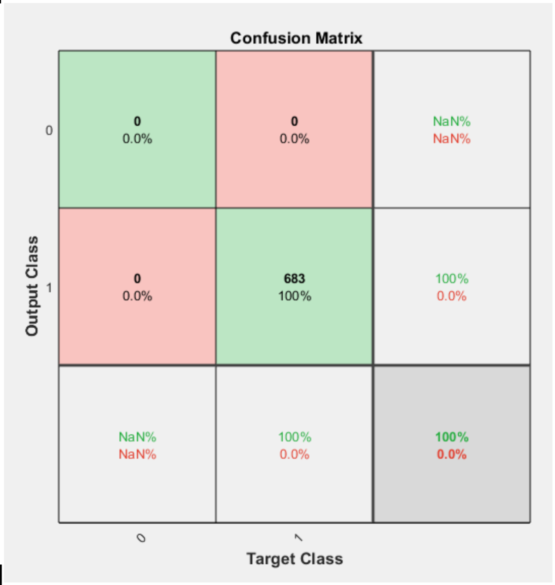
At the end of this fourth assignment, there are some points of view to analyze.

- In the Fig.1, the confusion matrices are optimal.
- In this representation there are not difference and there are not error. Maybe, I made some errors with the plot function, but I am not sure.
- In the Fig.2, I can see the output of the two neurons.
- I can appreciate that, when there are "easy" classes to distinguish, the dots are linearly separate, e.g the class 1 and 8. Instead, with "difficult" pairs the results are worse (Fig2.1 and Fig2.3).

I reported only few of the possible running. Up to now, I assume that results acceptable for this assignment.

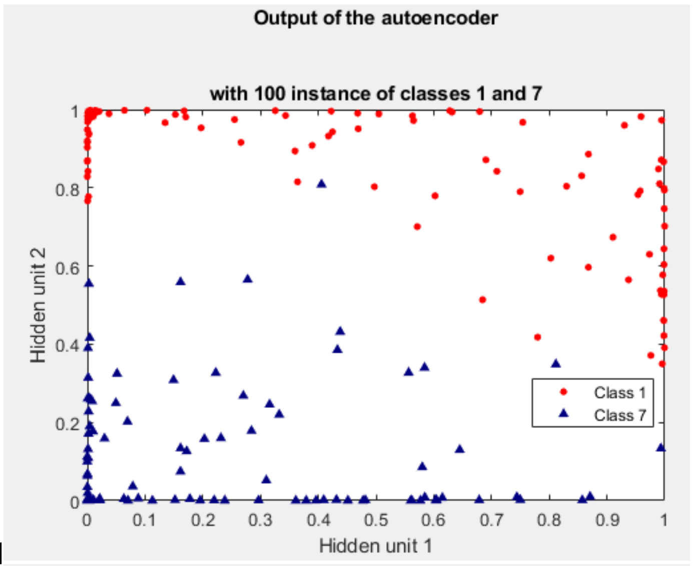


[a]

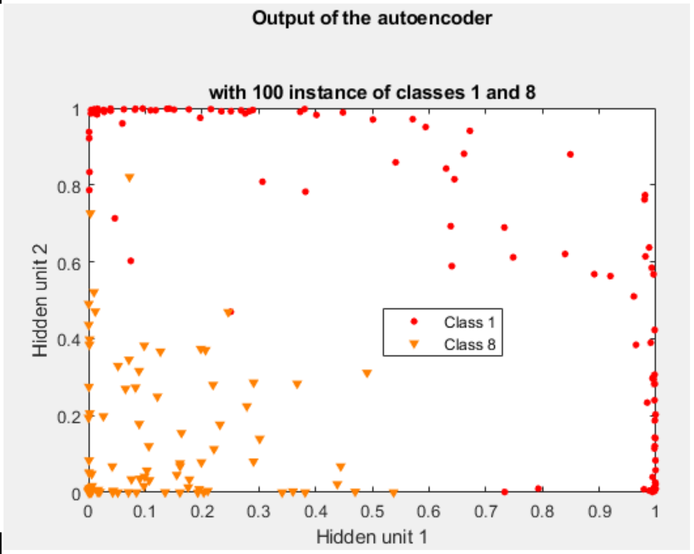


[b]

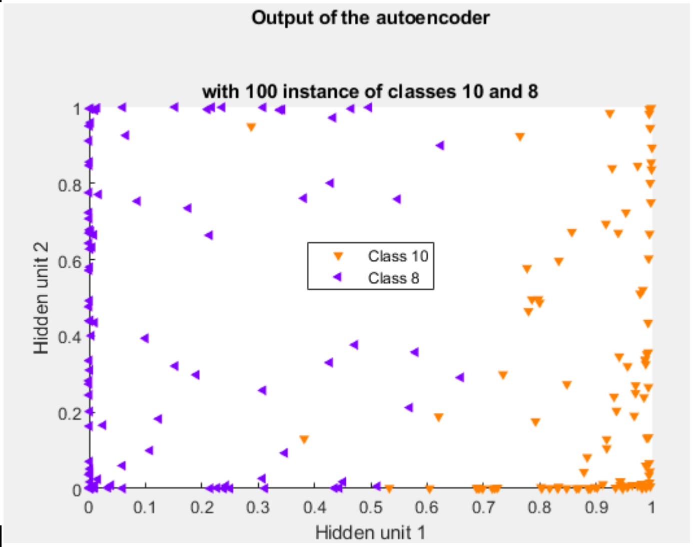
Fig. 2. CONFUSION MATRIX, represents the quality of the results.
(a) Wine set with 30 hidden units, with 10 size for each unit and *trainbr* method.
(b) BCW set with 30 hidden units, with 10 size for each unit and *trainbr* method.



[1]



[2]



[3]

Fig. 3. AUTO-ENCODER.
(1) classes 1 7.
(2) classes 1 8.
(3) classes 8 0(not 10).