Neural Networks

Machine Learning 2021/22: 4^{th}Assignment

Matteo Maragliano
EMARO-European Master on Advanced Robotics
DIBRIS - Dip. Informatica, Bioingegneria, Robotica, Ing. dei Sistemi
Università degli Studi di Genova

***Abstract*—The goal of the assignment is to test the Neural Network Toolbox of MATLAB. Thanks to this tool we could test and train different types of data sets and see results through *confusion matrices* or *ROC curve: Receiver Operative Characteristics curve* which describe the performances of the work.**
Data are often divided into classes: in our case the data set concerns the identification of cancer into benign or malignant using features of sample biopsies: in particular we had 9 inputs each one with 699 different observation and an output with 2 classes as just said before with 699 observations as well in order to classify the result.

I. Introduction: Neural Networks in MATLAB

NEURAL networks are networks of several, interconnected, simple units. They are a model for learning and mapping from inputs to outputs that tries to emulate the inner mechanics of brain processing.

The model of neural network used in our work is the following:

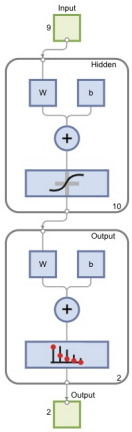

Fig. 1. Neural network model: as it can be seen in the figure there are 9 inputs (which represent 9 observations) and 2 outputs (which represent the 2 classes returned by the classification).

The basic element of the network is the neuron which is a simple unit that takes an input, makes a decision based on a function and returns an output that can be passes to another neuron or to the network itself. Formulas behind neurons is:

$$r = \mathbf{x} * \omega$$

$$f = (r - \theta)$$

where:

- x is the d-dimensional vector of input to the network and w are the corresponding weights;
- r is the net input on the neuron membrane;
- f is a nonlinear function used to take the decision;
- θ is a threshold;
- a is the output of the neuron (activation value or action potential).

There are various type of functions that can be used: some of the most popular are the *Heaviside Step*, the *signum* and the *sigmoid* represented by the follwing formula:

$$f(x) = \frac{1}{1 + e^{-x}}$$

II. Feedforward multi-layer networks (multi-layer perceptrons)

Inside the first task of the assignment we were supposed to learn from an article on how to use a tool.
The tool mentioned is the *Neural Network Fitting app* which is an application able to represent and simulate the behaviour of a neural network.
Thanks to this tool we could have an interface and so a better way to approach the problem. The model of the neural network proposed is the one showed in the figure, in particular the network is a two-layer feedforward network with a *sigmoid* transfer function in the hidden layer and a linear transfer function in the output layer. The *layer size* value defines the number of hidden neurons.
Using the application we chose a data set from a group proposed and trained the set; after that we had the result of the simulation in different forms: a model summary which contains information about the training algorithm and the training results for each data set; there is also the possibility of checking the results obtained by generating the graphs we want: confusion matrices, regressions, histograms and so on.
For our example we took the confusion matrices and the regression ones: the results obtained are shown in the following figures:

Model Summary

Train a neural network to classify predictors into a set of classes.

Data

Predictors: cancerInputs - [9x699 double]
Responses: cancerTargets - [2x699 double]

cancerInputs: double array of 699 observations with 9 features.
cancerTargets: double array of 699 observations with 2 classes.

Algorithm

Data division: Random
Training algorithm: Scaled conjugate gradient
Performance: Cross-entropy error

Training Results

Training start time: 13-Dec-2021 11:43:58
Layer size: 10

	Observations	Cross-entropy	Error
Training	489	0.0426	0.0286
Validation	105	0.0849	0.0381
Test	105	0.0897	0.0857

Fig. 2. Model Summary for the Neural Network with 70% of data for training, 15% of data for test and 15% of data for validation.

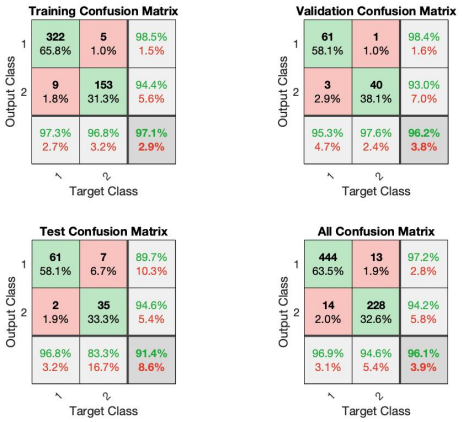

Fig. 3. Confusion Matrices for the Neural Network with 70% of data for training, 15% of data for test and 15% of data for validation.

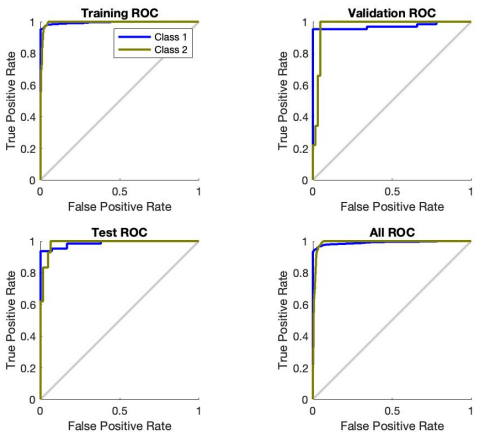

Fig. 4. ROC curve for the Neural Network with 70% of data for training, 15% of data for test and 15% of data for validation.

From the figure it is evident that the network outputs are very accurate, as it can be seen by the high numbers of correct classifications in the green squares (diagonal) and the low numbers of incorrect classifications in the red squares (off-diagonal).
Also from the other graph is evident what we jsut said, in particular the colored lines in each axis 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 this problem, the network performs very well.

III. Autoencoder

To finish our work we completed the task number 2 with the implementation of an *Autoencoder*.
The simplest autoencoder network is a multi-layer perceptron neural network which has one input layer, one hidden layer and one output layer; the model used is the following:

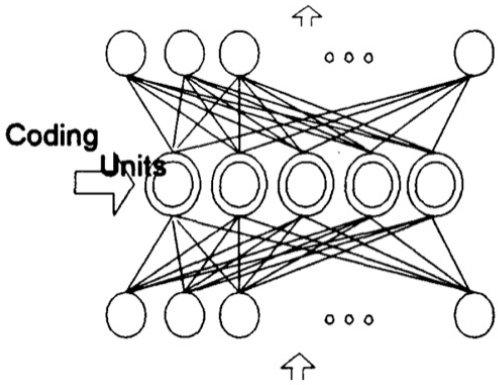

Fig. 5. Structure for the *Autoencoder* implemented.

It is trained using the same pattern as both the input and the target; note that in this case we don't have any classes or other mapping to learn. This is a special case of *unsupervised training*. In fact, it is sometimes called *self-supervised*, since the target used is the input pattern itself.
An autoencoder learns an internal, compressed representation for the data. The interesting output, therefore, is the value of its hidden layer. What we hope is that similar patterns will have similar representations, in other words, that the network will learn the classes.

To complete this, we gave 2 distinct classes as input and as train and we see the result obtained: what we expected was a quite clear split of the classes taken. After that we repeated the experiment with more classes passed and, as expected, the result was a bit worse than in the previous case: the division of the classes was not so clear and some items was not coded as well as before; this is a consequence of the amount of classes: more classes implicates a less precision in the classification.
The results obtained are shown in the following graphs:

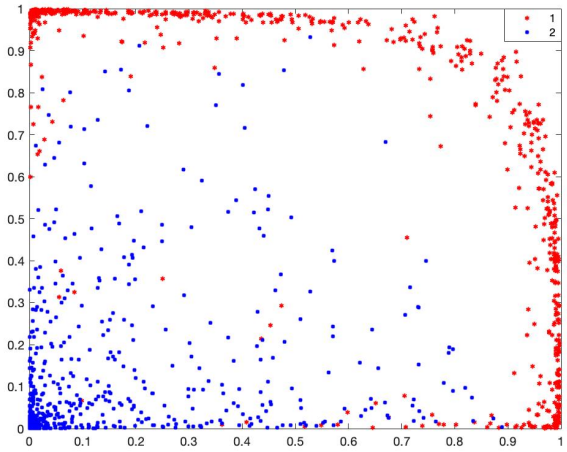

Fig. 6. Graphical representation of the autoencoder output with class 1 and 2 as input.

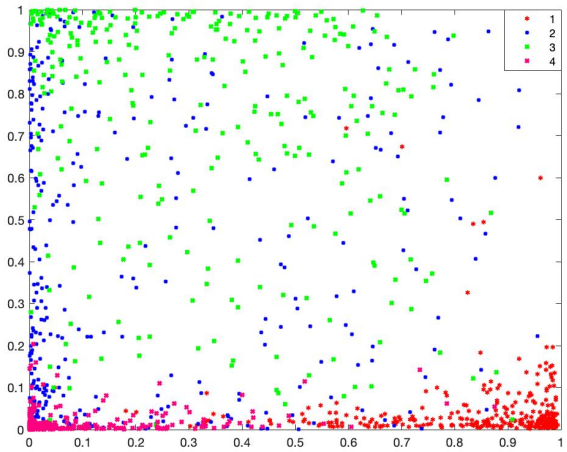

Fig. 7. Graphical representation of the autoencoder output with class from 1 to 4 as input.

As it can be clearly seen from the images above, the first result takes only 2 classes and the division is better than the second one, processed with 4 different classes.
In particular many items of one specific class are decoded as they would belong to another one, thus introducing a minor accuracy and of course a bigger error in the classification.

Moreover, there can be also some classifications with a small number of classes as input, two for example, that could be not so clear. The following figure makes it clearer:

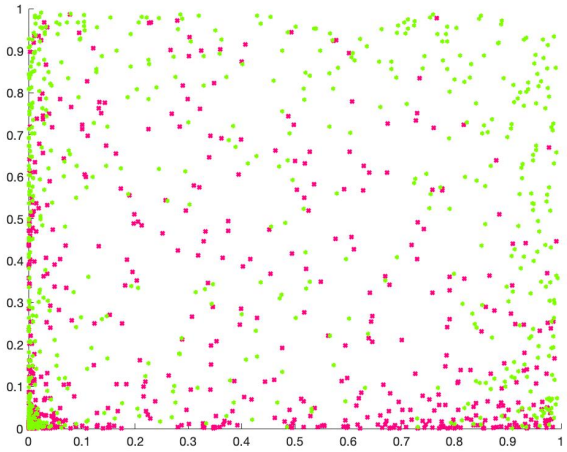

Fig. 8. Graphical representation of the autoencoder output with class from 4 and 9 as input

As said before, this is a case in which, even if there are only 2 classes, their division is not so clear as expected. This can be a result due to some similarities in the classes and in the figures provided as data set that can e confusing during the classification: we can imagine that the number *4* and *9* could be confused is written not so clearly so this can bring lots of error, as in the figure.