Università degli Studi di Genova

Robotics Engineering

Saivinay Manda

Report of Machine Learning course no.4

Neural Networks/ Auto encoders

Contents

Chapter 1

Introduction

This assignment deals with a single-unit neural networks: Perceptron and Adaline algorithms have been implemented through two MATLAB functions, then they have been tested. 4b deals with a multi-neural networks: MATLAB's own neural network tools is used. "Neural Networks Toolbox" is the MATLAB library which contains these tools. Two tutorials are followed: "Fit Data with a Neural Network" and "Classify Patterns with a Neural Network".

1.1 Neural Networks

(Artificial) Neural networks or ANNs are networks of several interconnected units. Each with a simple behaviour, most often without memory and capable of a single input-to-output transformation. There are usually several connections, and their properties are summarized in parameters called weights. The unit is called formal neuron and below the general scheme is showed.

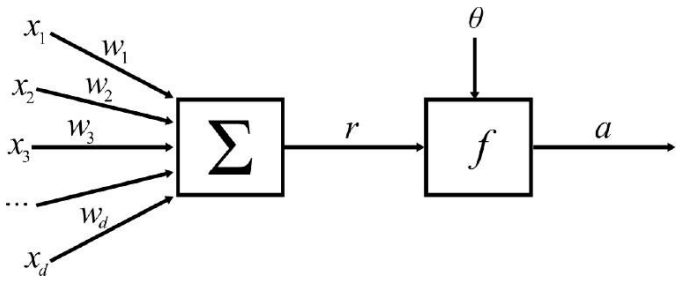

Figure 1.1: Models of neural functions

1) r is the net input;

2) f is the activation function;

3) a is the activation value.

In mathematical term we have:

$$r = x \cdot w$$

$$a = f(r - \vartheta)$$

x is a d-dimensional vector of input values w is the corresponding (d-dimensional) vector of synaptic weights, $\cdot$ indicates scalar product r indicates the net input on the neuron membrane f() is a non-linear function ϑ is a threshold a indicates the activation value of the membrane potential, or action potential; that is the output of the neuron.

As activation functions f, different types can be used:

1) Heaviside step;

2) Signum function;

3) Sigmoid and hyperbolic tangent;

4) Ramp;

5) ”Softplus” .

Learning in (formal) neurons occurs by slow modification of weights.

According to the steps t = 0,1,2,3,.., these is an iterative procedure that gradually changes weights until the delta rule goes to zero. The general formula is

$$w_{t+1} = w_t + \Delta w_{t+1}$$

1.1.1 Single-unit neural networks

Two simple models are presented:

1) Perceptron;

2) Adaline;

Perceptron Learning Algorithm

Perceptron is a linear classifier (binary). It helps to classify the given input data.

Learning with perceptron begins by taking all the input and multiplying them by their weights. Then, the weighted sum is created going to sum all multiplied values. The activation function plays the integral role of ensuring: the output is mapped between -1 and 1 (as well as 0 and -1).

The algorithm is:

(init) Perform initializations

1) INPUT: Read input pattern number l, x_l

2) Compute output: $r_l = w \cdot x_l$, $a_l = f(r_l)$

3) Compute output error: $\delta_l = 1/2(t_l - a_l)$

4) Compute correction: $\Delta w_l = \eta \delta_l x_l$

5) Apply correction (weight update): $w_{l+1} = w_l + \Delta w_l$

6) Increase l step

7) Go to INPUT.

An important concept to take into consideration is the **convergence**:

If the training set is not linearly separable, the procedure does not converge and the perceptron learning can not find a separating hyperplane for it in a finite number of steps.

Adaline Learning Algorithm

Adaline learning algorithm can be seen as an improvement of Perceptrton.

The main difference between Adaline and Perceptron is that the weights in Adaline are updated on a linear activation function rather than the unit step function as is the case with Perceptron.

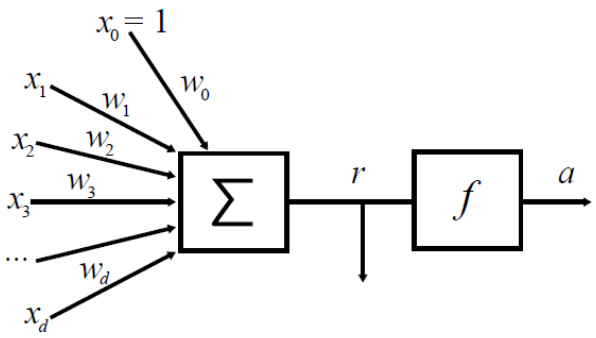

Figure 1.2: Adaline algorithm scheme

Notice that r is a continuous function of the weights, thus we have a quantitative evaluation of the error defined as the Sum of squared errors which becomes differentiable, and it is a convex shaped. The focus is to minimize the error during the training step. We can use the gradient descend to find the weights that minimize our cost function.

Thus, after calculation, we can update out weights following the rule:

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

where:

Delta rule is $\Delta w_l = \eta \delta_l x_l$

and $\delta = (t - r)$

Notice that $\delta_l x_l$ is just an estimate of the (negative) gradient based on one sample.

If η is small enough, this converges on average to the unique minimum.

For l goes to infinite, it will keep on oscillating, but always around the right solution.

1.1.2 Multi-layer networks

A multi-layer neural network is composed of:

a) an input layer;

b) one (or more) hidden layer;

c) an output layer, that provides the output of the network.

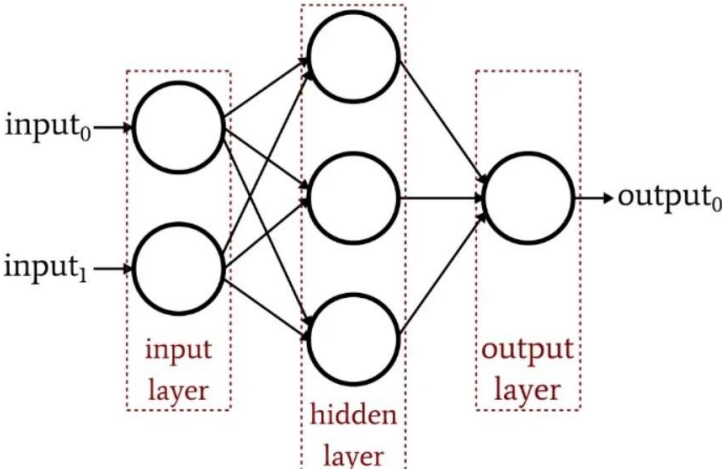

Figure 1.3: General scheme of a multi-layer network

Learning is a sort of optimization of the cost function witch permits to minimize the error. Most optimization methods are based on the gradient descent, with requires that f must be at least differentiable.

In the multi-layer network, even if f is differentiable, for hidden units we cannot compute an error term.

A possible solution to find this error is: using a differentiable activation function and compute the gradient for all layers with the error back-propagation algorithm.

An important consideration should be given to the number of hidden layer we want to use: in artificial neural networks, hidden layers are required if and only if the data must be separated non-linearly.

The optimal number of hidden units could easily be smaller than the number of inputs, there is no rule like multiply the number of inputs with N,... Usually it is used one hidden layer for simple tasks, but nowadays research in deep neural network architectures show that many hidden layers can be fruitful for difficult object, handwritten character, and face recognition problem.

Chapter 2

Lab 4: Neural Network

The goal of this assignment was to use MATLAB's own neural network tools, contained in a library called Neural Networks Toolbox. The work was composed by two tasks:

1) Task 0: Execute Matlab tutorial: "Fit Data with a Neural Network" to get acquainted with the toolbox;

2) Task 1: Execute the tutorial "Classify Pattern with a Neural Network", using different sets and different values of the parameters.

2.1 Results

For all tasks, the used dataset are contained in MATLAB toolbox: the structure of the dataset has already columns and rows ordered according to the MATLAB neural network toolbox. For opening the MATLAB GUI, the command "nnstart" has been used.

The tools themselves guide you through the process of designing neural networks to solve problems in four important application areas, without requiring any background.

In addition, the tools can automatically generate both simple and advanced MATLAB scripts that can reproduce the step performed by tool.

2.1.1 Task 0

The focus in task 0 is to define a fitting problem, using the Chemical dataset in which the number of observation are 498, with 8 attributes and 1 target.

The chosen training algorithm was Levenberg-Marquardt and 70 % of the overall dataset is used for training, 15 % for validation and the other 15 % for testing (the splitting was executed randomly).

The number of hidden layer used is 10, and the results look good, with a small error: increasing the number of hidden layer the results, of course, will improve.

The error histogram is showed, in which the blue bars represent training data, the green bars represent validation data, and the red bars represent testing data. The histogram can give you an indication of outliers, which are data points where the fit is significantly worse than the majority of data.

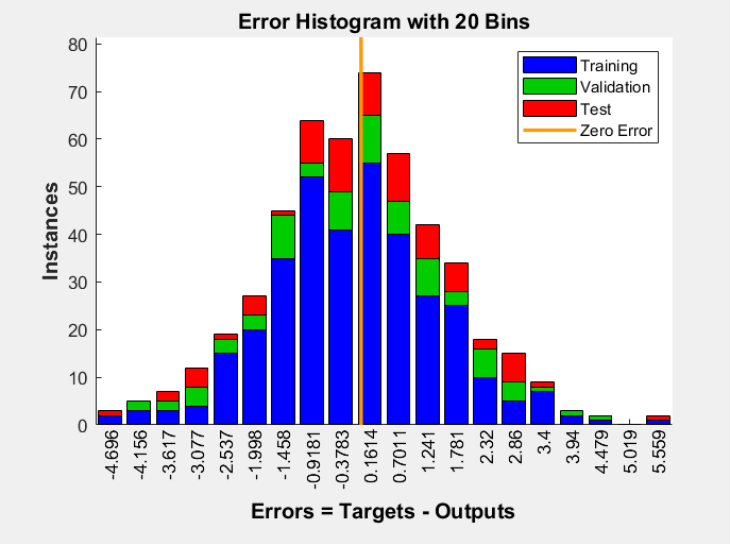

Figure 3.1: Error Histogram Fitting problem with Hidden Layer = 10

To improve the performance and to decrease the range in which most errors fall, we can increase the number of the hidden Layer and train again.

If the performance on the training set is good, but the test set performance is significantly worse, which could indicate overfitting, then reducing the number of neurons can improve your results. If training performance is poor, then you may want to increase the number of neurons.

2.1.2 Task 1

The focus in task 1 is to define a Pattern recognition, using two dataset: Wine dataset and Glass dataset.

Also in this case, 70 % of the overall dataset is used for training, 15 % for validation and the

other 15 % for testing (the splitting was executed randomly).

Some Confusion matrices are plotted to check the accuracy in several cases.

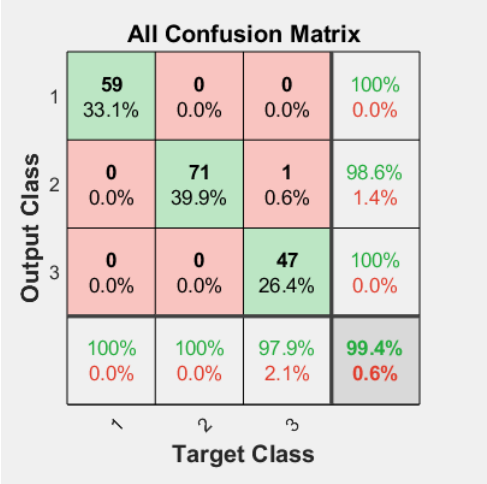

Figure 2.2: Confusion matrix of the Wine Dataset with Hidden Layer = 10

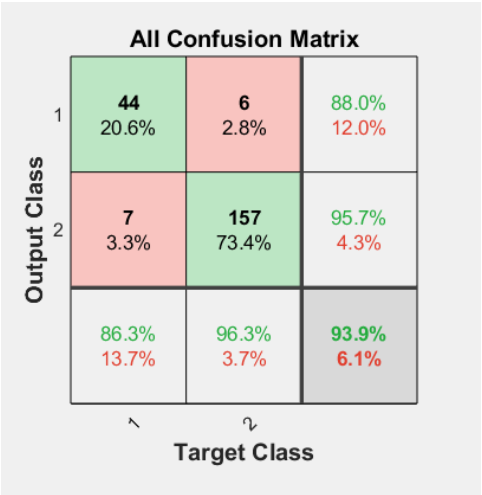

Figure 2.3: Confusion matrix of the Glass Dataset with Hidden Layer = 10

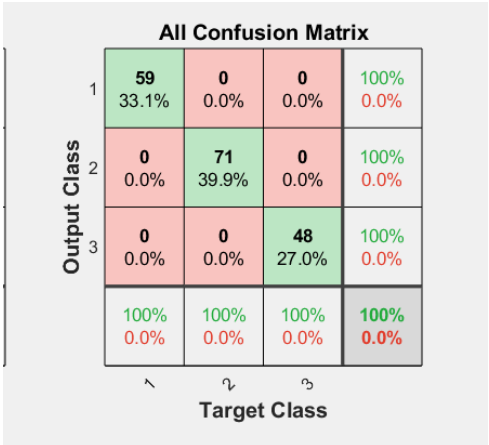

Figure 2.4: Confusion matrix of the Wine Dataset with Hidden Layer = 50

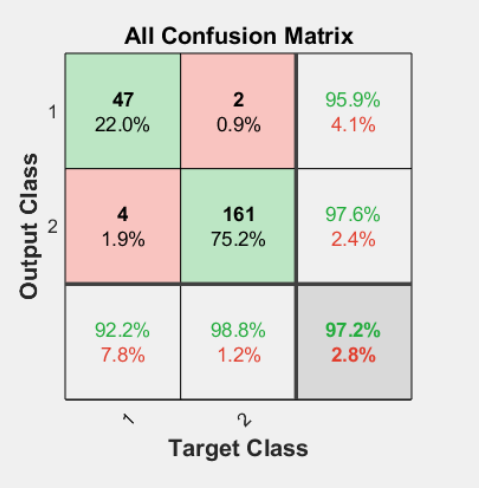

Figure 2.5: Confusion matrix of the Glass Dataset with Hidden Layer = 50

As we can notice from the figure, increasing the number of hidden layers, the accuracy improves.

Chapter 3

Lab 4: Autoencoder

To execute the first part of the lab, consider as input the MNIST dataset, composed of images handwritten digits from 0 to 9. Considering the large dataset, we had to select a subset composed of only two digits with reduced dimension (500 randomly samples for each). The experiment is done with different combinations (some will be easier to learn (for instance 1 and 8), some harder, (for instance 1 and 7). To train and encode the dataset, two made Matlab funcions are used: ”trainAutoencoder()” and ”encode()”; The num of units for the hidden layer is 2, in such a way we can have 2-dimensional plot. The output is the value of its hidden layer: we hope that similar patterns will have similar representations. To plot the results, we had to use the given ”plotcl()” function.

3.1 Results

The evaluation of the task was done for different combinations of two digits, considering 2 units of the hidden layer: indeed in the following plots, we have got two different colors point which represent two neurons.

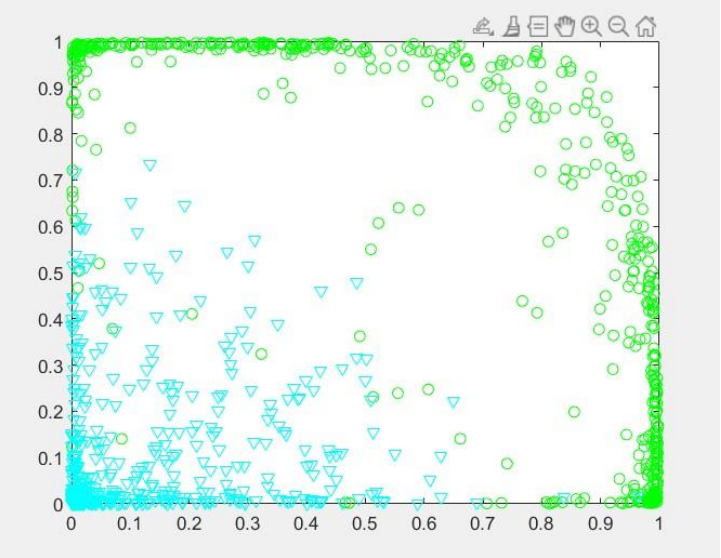

Figure 3.1: Digits 1 and 8

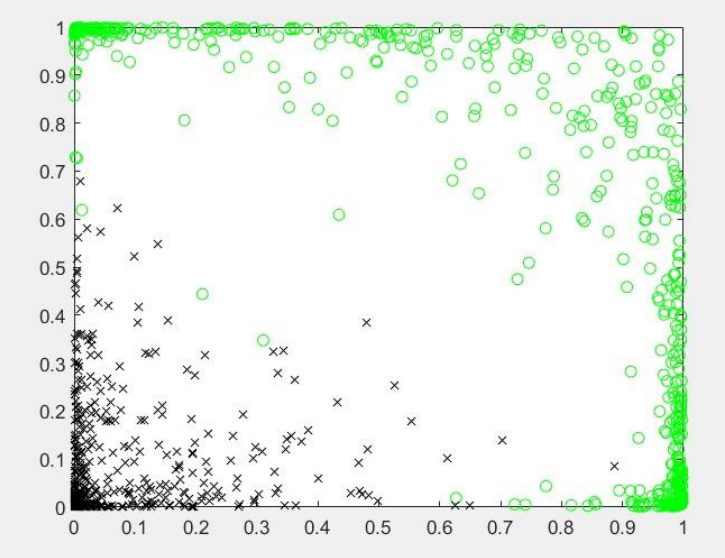

Figure 3.2: Digits 1 and 4

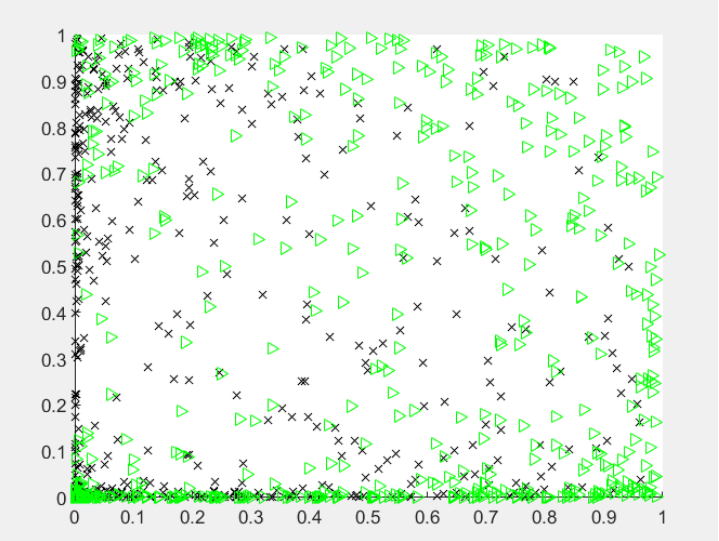

Figure 3.3: Digits 4 and 9

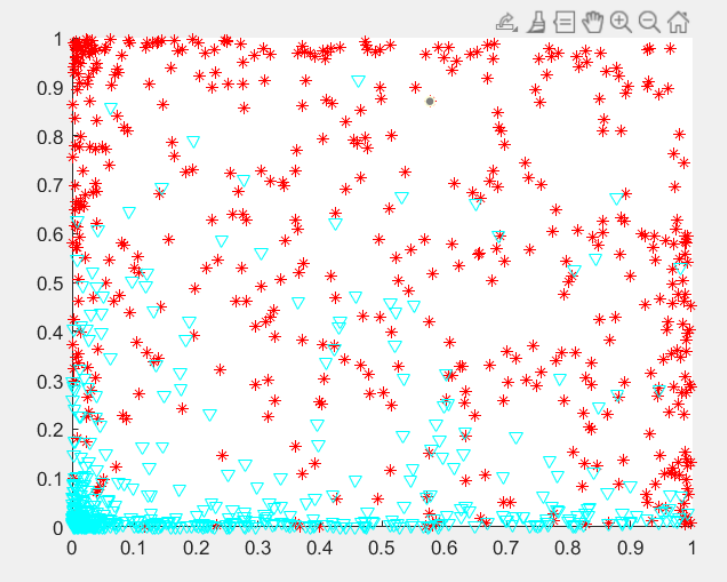

Figure 3.4: Digits 3 and 8

As we can see, if the shapes of two digits are very different to each other, then the different points are easily linearly separable (for instance 1-8 and 1-4), otherwise, the points are mixed, and to separate them results hard (for instance 4-9 and 3-8).

Applied Deep Learning - Part 3: Autoencoders 2017

References

Adaptive linear neuron (2020). Arjun's Blog. URL: https://arjunkathuria.com/ml/Adaline/.

Beginners Ask "How Many Hidden Layers/Neurons to Use in Artificial Neural Networks? (2018). Towards data science. URL: https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e.

Classify Patterns with a Shallow Neural Network (2018). Mathworks. URL: https://it.mathworks.com/help/deeplearning/gs/classify-patterns-with-a-neural-network.html.

Fit Data with a Shallow Neural Network (2018). Mathworks. URL: https://it.mathworks.com/help/deeplearning/gs/fit-data-with-a-neural-network.html.

https://2020.aulaweb.unige.it/pluginfile.php/276344/mod_folder/content/0/ml-2020-21--07.pdf?forcedownload=

Shallow Networks for Pattern Recognition, Clustering and Time Series (2018). Mathworks. URL: https://it.mathworks.com/help/deeplearning/gs/shallow-networks-for-pattern-recognition-clustering-and-time-series.html.

Applied Deep Learning - Part 3: Autoencoders (2017). Towards data science. URL: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798