Machine learning Assignment4

Chiara Saporetti

November 2020

Abstract This assignment is divided in two parts:
Task A aims at implementing in MATLAB two simple single-layer neural networks. Specifically they are the perceptron and the adaline algorithms.
Task B addresses more complex cases and aims at getting acquainted with Matlab built-in functions and GUI for neural networks.

1 Introduction

Adaline and Perceptron are two examples of single-unit neural networks [4]. They are by-pattern algorithms, in which weight modifications are computed upon receiving each individual pattern. We assume a linear threshold unit with activation function f(r)=sign(r). The basic implementation is represented in Figure 1: all inputs are weighed and summed together before being multiplied for the activation function.

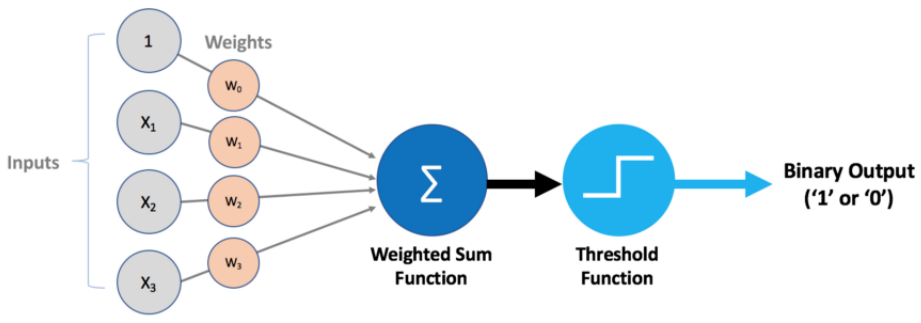


Figure 1: Basic algorithm model

1.1 Perceptron algorithm

The model is represented in Figure 2: the idea is the same as before, but the output of the threshold is used as a feedback value. If the training set is linearly

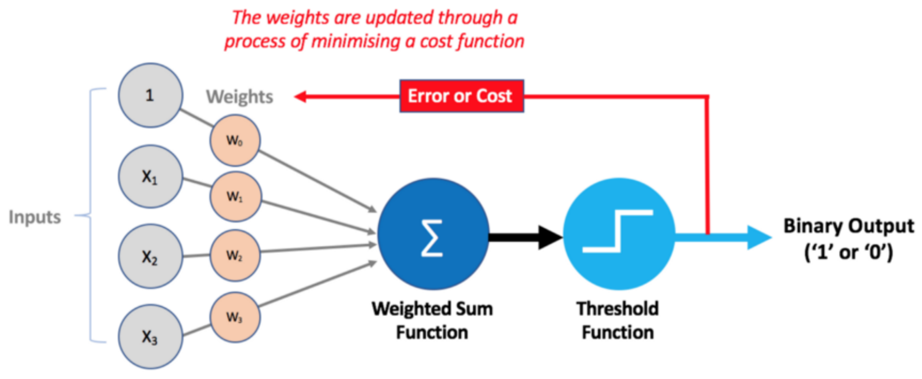


Figure 2: Perceptron algorithm model

separable, the perceptron learning procedure will find a separating hyperplane for it in a finite number of steps. If not, the procedure doesn't converge.
In pseudo code:

```
w=rand(0,1)
xl=1
for count=1:10000
    r=x(xl)*w
    a=sign(r)
    delta=0.5*(t(xl)-a)
    w=w+eta*delta*x(xl)
    xl++
```

1.2 Adaline algorithm

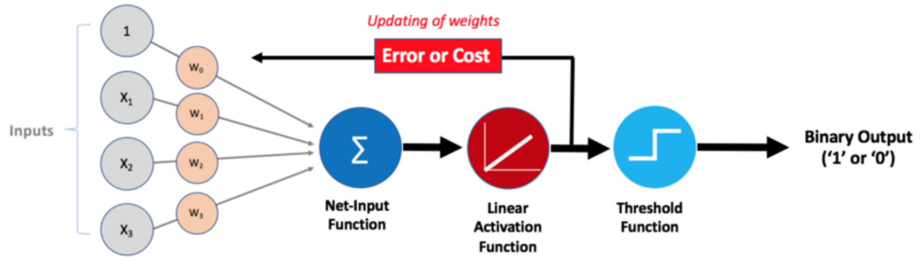


Figure 3: Adaline algorithm model

The adaline algorithm is a perceptron in which, in addiction to a, also r is available outside. It is better than the perceptron in the sense that it converges even for non-linearly separable problems and achieves more robust solutions. The model is represented in Figure 3 In pseudo code:

```
w=rand(0,1)
xl=1
for count=1:10000
    r=x(xl)*w
    delta=0.5*(t(xl)-r)
    if delta<threshold
        break
    w=w+eta*delta*x(xl)
    xl++
```

1.3 Confusion matrix

The results of both algorithms are shown as confusion matrices. This is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (like in this case). Each row of the matrix represents the instances in the predicted class while each column represents the instances in the actual class. This means that the right guesses areon the main diagonal, so it is here that we expect to see higher values.

2 Task A

2.1 The datasets

- **MNIST dataset:** Dataset composed of images of handwritten digits from 0 to 9.
  - Examples in training set: 600000.
  - Examples in test set: 100000.
  - Classes: 10.
- **Iris dataset:** Reduced version of the original dataset. The original version contains a set of 150 records under five attributes - sepal length, sepal width, petal length, petal width and species. In this example it is modified to have only two linearly-separable classes.
  - Examples in data set: 150.
  - Classes: 2.
- **XOR dataset:** Simple dataset that comes from the truth table of the XOR.
  - Examples in data set: 8.
  - Classes: 2.

2.2 Matlab implementation

2.2.1 main.m

The main loads the three datasets and makes them compliant with the rest of the code. In particular:

- For the iris dataset: it transforms the labels values into ± 1.
- For the MNIST dataset: it chooses two digits to recognize and sets their labels to ± 1. The other examples are removed.
- For the XOR dataset: it transforms the labels values into ± 1.

Then it calls the functions in order to perform the tasks.

2.2.2 splitDataSet.m

Input:

- x: original dataset
- k: number of training + test set couples

Output:

- x_res: structure containing the training sets and test sets
- num_sets: number of generated sets

This function splits a given dataset in n couples of training set + test set. The number n depends on the k value:

- if k=2: n=1. It splits the set in 2 equal parts, thus generating one couple only.
- if k=ROWS: n=ROWS. Where ROWS indicates the number of examples of the set. It performs leave-one out cross validation. This means that it splits the sample into ROWS training sets of size ROWS-1 and corrensponding ROWS test sets of size 1.
- if 2 <k <ROWS: n=k. It performs k-fold cross validation. This means that it splits the sample into k training sets of size ROWS/k and corrensponding test sets of size ROWS-ROWS/k.

2.2.3 perceptron.m

Input:

- x_total: structure with n training sets and n test sets
- eta: learning rate

- num_sets: how many training+test sets are given in x_total

Output:

- myconfusionMatrix: confusion matrix
- w: weight matrix

This function implements the perceptron algorithm, and returns the confusion matrix and the weight matrix.
In detail: it enters a loop and, at each step, takes a different couple of training and test sets. It generates a random vector or weights, sets a desired learning rate and a threshold to which δ will be compared to. Then it starts a for loop of ten thousands iterations. At each of them, starting from xl=1, it calculates r, a and δ and updates the weight. Then it increases xl and starts again.
At the end of the loop it has computed the desired set of weights and it uses it in the testing phase. It predicts the labels of the test set and then compares them to the real ones and builds the relative confusion matrix. After completing the for loop it averages the matrix and sets it in output.

2.2.4 adaline.m

Input:

- x_total: structure with n training sets and n test sets
- eta: learning rate
- num_sets: how many training+test sets are given in x-total

Output:

- myconfusionMatrix: confusion matrix
- w: weight matrix

This function implements the adaline algorithm, and returns the confusion matrix and the weight matrix.
In detail: it enters a loop and, at each step, takes a different couple of training and test sets. It generates a random vector or weights, sets a desired learning rate and a threshold to which δ will be compared to. Then it starts a for loop of ten thousands iterations. At each of them, starting from xl=1, it calculates r, a and δ, then checks the stopping condition: if δ is lower than a threshold it exits. It finally updates the weight, increases xl and starts again.
At the end of the loop it has computed the desired set of weights and it uses it in the testing phase. It predicts the labels of the test set and then compares them to the real ones and builds the relative confusion matrix. After completing the for loop it averages the matrix and sets it in output.

2.2.5 images.m

This function simply creates the tables of the confusion matrices and saves them as images.

2.3 Results

By considering a small learning rate we obtained the following results.
The perceptron algorithms works well on both the Iris (Figure 5) and the MNIST (Figure 6) datasets, obtaining less than 2% of errors in classification in all cases. In general, it performs better with k=2.
The Adaline algorithm seems to perform worse than the Perceptron but still obtains good results (Figures 7 and 9).
The XOR dataset was used to study the difference in convergence of the two algorithms. The XOr, or "exclusive or", problem is a classic problem in artificial neural networks research. It is the problem of using a neural network to predict the outputs of XOr logic gates given two binary inputs. The problem with these data is that XOr inputs are not linearly separable [3] . This is particularly visible by plotting the XOr input values to a graph, as in figure 4. The results obtained were the same for Perceptron and Adaline, and they showed that neither of them can cope with the non linearly separability case.

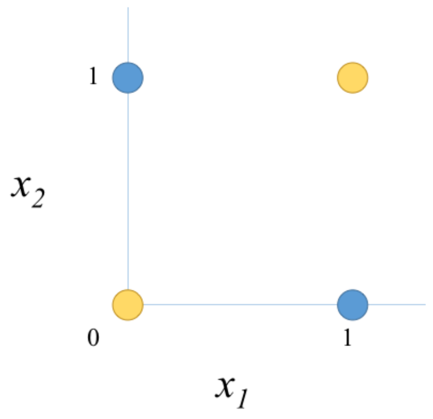
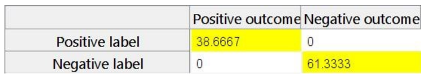
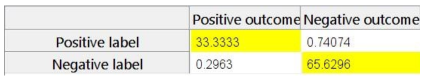


Figure 4: XOR inputs



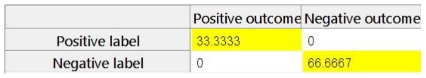
	Positive outcome	Negative outcome
Positive label	38.6667	0
Negative label	0	61.3333

(a) k=2



	Positive outcome	Negative outcome
Positive label	33.3333	0.74074
Negative label	0.2963	65.6296

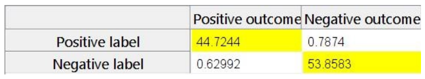
(b) k=10



	Positive outcome	Negative outcome
Positive label	33.3333	0
Negative label	0	66.6667

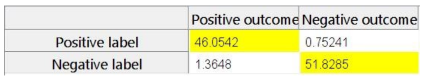
(c) k=150

Figure 5: Perceptron on iris dataset



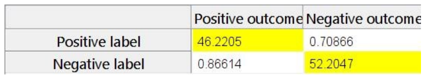
	Positive outcome	Negative outcome
Positive label	44.7244	0.7874
Negative label	0.62992	53.8583

(a) k=2



	Positive outcome	Negative outcome
Positive label	46.0542	0.75241
Negative label	1.3648	51.8285

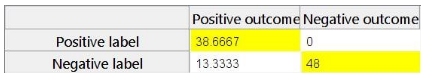
(b) k=10



	Positive outcome	Negative outcome
Positive label	46.2205	0.70866
Negative label	0.86614	52.2047

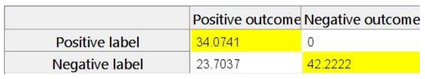
(c) k=150

Figure 6: Perceptron on MNIST dataset



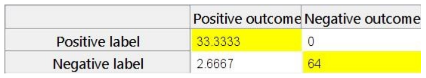
	Positive outcome	Negative outcome
Positive label	38.6667	0
Negative label	13.3333	48

(a) k=2



	Positive outcome	Negative outcome
Positive label	34.0741	0
Negative label	23.7037	42.2222

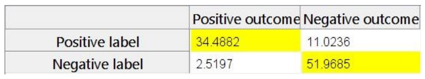
(b) k=10



	Positive outcome	Negative outcome
Positive label	33.3333	0
Negative label	2.6667	64

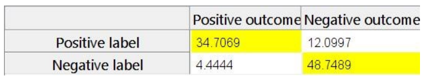
(c) k=150

Figure 7: Adaline on iris dataset



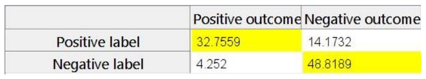
	Positive outcome	Negative outcome
Positive label	34.4882	11.0236
Negative label	2.5197	51.9685

(a) k=2



	Positive outcome	Negative outcome
Positive label	34.7069	12.0997
Negative label	4.4444	48.7489

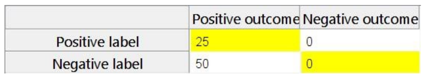
(b) k=10



	Positive outcome	Negative outcome
Positive label	32.7559	14.1732
Negative label	4.252	48.8189

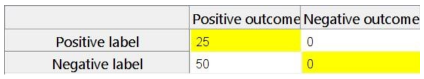
(c) k=150

Figure 8: Adaline on MNIST dataset



	Positive outcome	Negative outcome
Positive label	25	0
Negative label	50	0

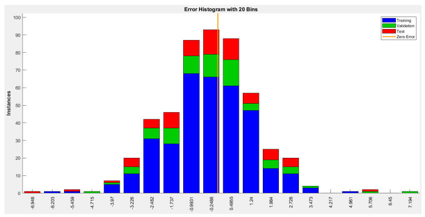
(a) k=2



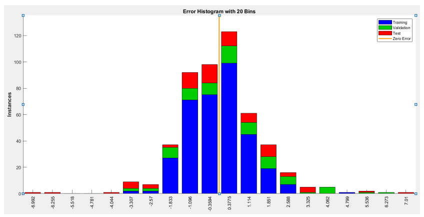
	Positive outcome	Negative outcome
Positive label	25	0
Negative label	50	0

(b) k=10

Figure 9: Perceptron and Adaline on XOR dataset



(a) Chemical dataset, 10 hidden units



(b) Chemical dataset, 20 hidden units

Figure 10: Error histogram

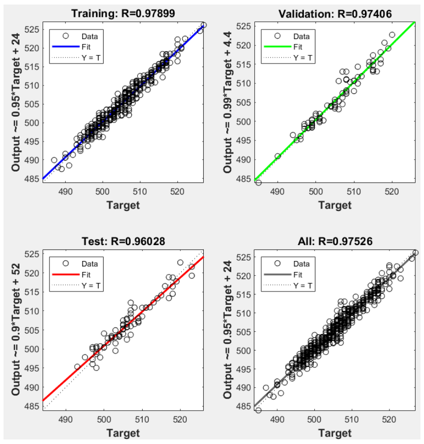
3 Task B

3.1 Fitting functions

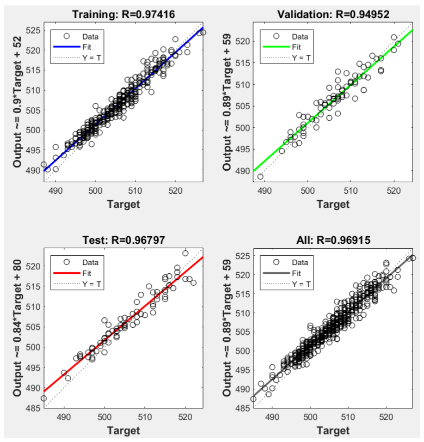
This task is about following Matlab's tutorial on neural networks used to find fitting functions for different datasets: [2]. Here are shown the results obtained on the chemical dataset. Figure 10 represents the error histograms of the chemical dataset for different values of hidden layers. The error is computed by computing the difference between output ans target.
Figure 11 represents the regression plots of the chemical dataset for different values of hidden layers. The small circles represent data, the line indicates the fitting line, the dotted line indicates Y=T, where Y is the output, and T is the target.

3.2 Pattern recognition

This task is about following Matlab's tutorial on neural networks used to classify patterns: [1].
Figure 10 represents the confusion matrices of the iris dataset for different values of hidden layers. On the main diagonal of the matrix (in green) are shown the correct classifications. By increasing the number of hidden units the results are generally imporved.

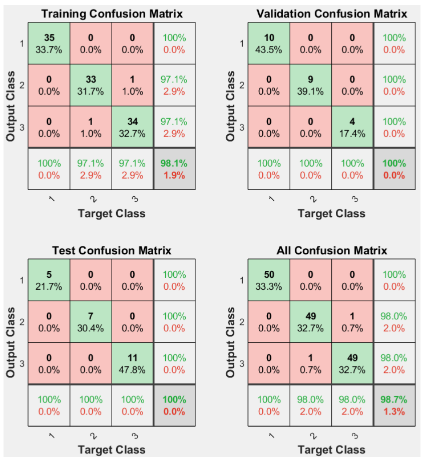


(a) Chemical dataset, 10 hidden units

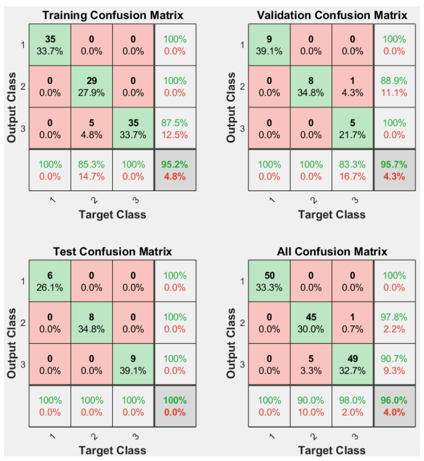


(b) Chemical dataset, 20 hidden units

Figure 11: Regression plot



(a) Iris dataset, 10 hidden units



(b) Iris dataset, 20 hidden units

Figure 12: Confusion matrix

References

[1] *https://it.mathworks.com/help/deeplearning/gs/classify-patterns-with-a-neural-network.html.*

[2] *https://it.mathworks.com/help/deeplearning/gs/fit-data-with-a-neural-network.html.*

[3] *https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b.*

[4] *https://towardsdatascience.com/understanding-basic-machine-learning-with-python-perceptrons-and-artificial-neurons-dfae8fe61700.*