



# Universidade Federal do Ceará

Disciplina: Inteligência Computacional Aplicada

Professor(a): Guilherme Barreto and Ajalmar

Estudante: Rubem Vasceconcelos Pacelli

Matrícula: 519024

Junho/2022

## Neural network - Report

### 1 Work 02 - Rosenblatt's perceptron

This work considers a binary classification problem of the iris flower dataset by using Rosenblatt's perceptron. The problem consists in classifying Setosa specie from the dataset.

Rosenblatt's perceptron comprises a neuron mathematical model, introduced by McCulloch and Pitts in 1943, and a learning algorithm that adjusts the synaptic weights in a supervised fashion. The McCulloch and Pitts' activation function is a step function that triggers the output from 0 to 1 when the induced local field overpasses the threshold. This method is effective for binary classifications of linearly separable problems, where one can sketch a straight line that divides the classes without overlapping.

At the instant  $n$ , the induced local field is given by

$$v(n) = \mathbf{w}^T(n) \mathbf{x}(n), \quad (1)$$

where

$$\mathbf{w}(n) = \begin{bmatrix} w_0(n) & w_1(n) & \cdots & w_{N_a}(n) \end{bmatrix}^T \quad (2)$$

and

$$\mathbf{x}(n) = \begin{bmatrix} x_0(n) & x_1(n) & \cdots & x_{N_a}(n) \end{bmatrix}^T \quad (3)$$

are the synaptic weights of the perceptron and the input signal, respectively, and  $N_a$  indicates the number of attributes. The elements  $w_0(n)$  and  $x_0(n) \triangleq +1$  are, respectively, the bias and its dummy input.

Let us first derive the update equation for Rosenblatt's perceptron in batch processing and then we take the sequential processing as a special case of it. We can define the

---

<sup>1</sup>Depending on the author, it can be defined as  $-1$ .

perceptron cost function as

$$\mathcal{E}(\mathbf{w}(n)) = - \sum_{\mathbf{x}(n) \in \mathcal{X}} \mathbf{w}^\top(n) \mathbf{x}(n) d(n) \quad (4)$$

Where  $\mathcal{X}$  is the set of all input signal  $\mathbf{x}(n)$  of the batch dataset that was misclassified by using  $\mathbf{w}(n)$ , and  $d(n) \in \{-1, 1\}$  is the desired signal with antipodal encoding. Since  $\mathcal{E}(\mathbf{w}(n))$  is differentiable, we can take this derivative with respect to  $\mathbf{w}(n)$ , yielding the gradient vector, that is,

$$\mathbf{g} = \frac{\partial \mathcal{E}(\mathbf{w}(n))}{\partial \mathbf{w}(n)} = - \sum_{\mathbf{x}(n) \in \mathcal{X}} \mathbf{x}(n) d(n) \quad (5)$$

The gradient (or steepest) descent method used the opposite direction of the gradient vector as the learning equation, i.e.,

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \mathbf{g} \quad (6)$$

$$= \mathbf{w}(n) + \eta \sum_{\mathbf{x}(n) \in \mathcal{X}} \mathbf{x}(n) d(n) \quad (7)$$

where  $\eta$  is the learning rate hyperparameter. For the particular case of sequential processing, the set  $\mathcal{X}$  becomes the  $\mathbf{x}(n)$  only in case of error. Otherwise,  $\mathcal{X}$  is an empty set. Therefore, we can rewrite<sup>2</sup> (6) as

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta e(n) \mathbf{x}(n), \quad (8)$$

where  $e(n) = d(n) - y(n) \in \{-2, 0, 2\}$  is the error signal. The Equation (1) passes through the step function,  $\varphi(\cdot)$ , generating the perceptron output,  $y(n) = \varphi(v(n)) \in \{-1, 1\}$ . This signal is compared to the desired value and produces the error signal, which indicates whether the perceptron is misclassified or not.

The Algorithm 1 summarizes the procedure utilized for Rosenblatt's perceptron. It includes data preparation techniques, such as hand-out, data shuffling, and data standardization. The method utilizes  $N_r = 20$  independent realizations and passes through the training set by  $N_e = 100$  epochs. At the end of each realization, it is stored the accuracy<sup>3</sup> reached by the test dataset, and the accuracy of all realizations is investigated in terms of mean and standard deviation. The iris dataset contains  $N = 150$  instances with  $N_a = 4$  attributes (petal length, petal width, sepal length, and sepal width) and  $K = 2$  classes (Setosa and nonSetosa). It was chosen a ratio of 80% – 20% for the training and

---

<sup>2</sup>Ignoring the double factor inserted by  $e(n) \in \{-2, 0, 2\}$

<sup>3</sup>Accuracy is defined as the ratio of the number of correct predictions by the total number of predictions.

test datasets, respectively <sup>4</sup>.

---

**Algorithm 1:** Rosenblatt's perceptron

---

```

Input:  $\mathbf{X}, \mathbf{d}$  // attributes and labels dataset
1 forall  $\{1, 2, \dots, N_r\}$  do
2    $\mathbf{w}(n) \leftarrow \text{initialize}$ 
3    $\mathbf{X}, \mathbf{d} \leftarrow \text{shuffle}$ 
4    $\mathbf{X}, \mathbf{d} \leftarrow \text{standardize}$ 
5    $(\mathbf{X}_{trn}, \mathbf{d}_{trn}), (\mathbf{X}_{tst}, \mathbf{d}_{tst}) \leftarrow \text{hold-out}$  // training and test dataset
6   forall  $\{1, 2, \dots, N_e\}$  do
7     forall Instances in the training dataset do
8        $v(n) \leftarrow \mathbf{w}^\top(n) \mathbf{x}(n)$ 
9        $y(n) \leftarrow \varphi(v(n))$ 
10       $e(n) \leftarrow d(n) - y(n)$ 
11       $\mathbf{w}(n+1) \leftarrow \mathbf{w}(n) + \eta \mathbf{x}(n) e(n)$ 
12     $\mathbf{X}_{trn}, \mathbf{d}_{trn} \leftarrow \text{shuffle}$ 
13     $accuracy \leftarrow \text{test}(\mathbf{X}_{tst}, \mathbf{d}_{tst})$ 
14 Compute the mean and the standard deviation of all accuracies.

```

---

The process described in Algorithm 1 was repeated for each class and results are shown in Table 1. The setosa class clearly outperforms other classes since it is linearly separable for some attributes, as shown in the decision surface in Figure 1<sup>5</sup>.

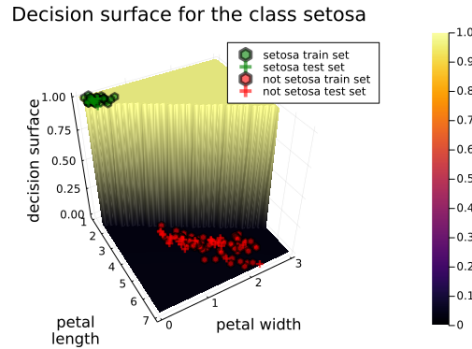


Figure 1: Decision surface of setosa class.

The confusion matrix of the setosa class is shown in Figure 2 for the first realization. The main diagonal indicates that there were neither false negatives nor false positives.

---

<sup>4</sup>The values of  $N_e$ ,  $N_r$ , and the train-test ratio are maintained throughout this homework.

<sup>5</sup>Since the problem has four attributes, this plot would be impossible as we would get 2 degrees of freedom. Therefore, for this result, we considered only the two attributes shown in this figure.

Table 1: Rosenblatt’s perceptron performance for classification problem

Classes	mean accuracy	standard deviation
Setosa	98.33	0.01972
Virginica	54.16	0.1251
Versicolor	53.66	0.1591

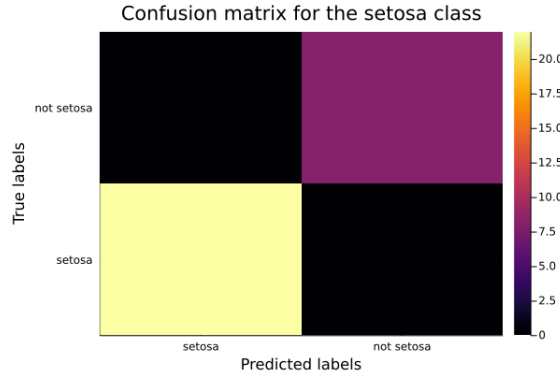


Figure 2: confusion matrix for setosa class.

The Figure 3 shows the evolution of the training dataset accuracy throughout the epochs for a given realization. One can notice the fast convergence to the accuracy of 100%.

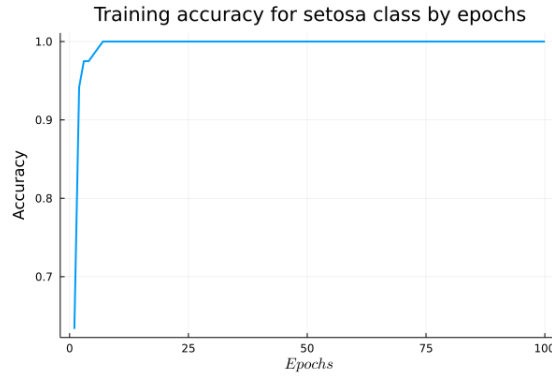


Figure 3: Training dataset evolution for the setosa classification.

For a dummy dataset with  $K = 4$  classes, the Rosenblatt’s perceptron achieved a mean accuracy of 100%. The Figure 4 shows the decision surface of the desired class for the realization whose accuracy is the closest to the mean accuracy. All instances of all classes are samples drawn from a Gaussian distribution with a given mean and variance.

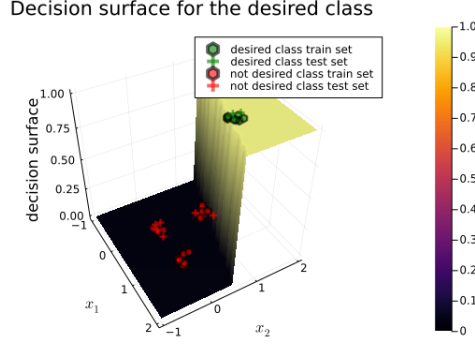


Figure 4: Decision surface for the desired class.

## 2 ADALINE

The Adaptive Linear Element (or ADALINE) is a variation of the Rosenblatt's perceptron, where the step function is replaced by a linear function, that is,  $y(n) = \varphi(u(n)) = u(n)$ . One can combine a tapped delay line with an ADALINE, thus creating an adaptive filter, widely used in statistical signal processing.

Consider a regression problem where the desired signal comes from a function  $f(x)$  corrupted with Gaussian noise. The ADALINE model tries to retrieve the original data using the same process described in Algorithm 1. However, the performance analysis uses the MSE error instead the accuracy since it is now a regression problem.

The Table 2 shows the performance of the mean MSE and its standard deviation obtained over independent realizations, in addition to the root mean squared error (RMSE). We consider scenarios where  $f(\cdot)$  is a function of one or two variables. In other words, for the first scenario, the input is the vector

$$\begin{bmatrix} 1 & x(n) \end{bmatrix} \in \mathbb{R}^2 \quad (9)$$

and  $f_1(x) = ax(n) + b$ , while the input vector for the second scenario is given by

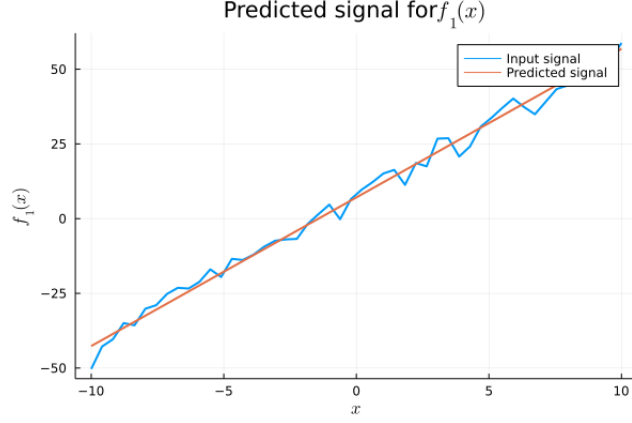
$$\begin{bmatrix} 1 & x_1(n) & x_2(n) \end{bmatrix} \in \mathbb{R}^3 \quad (10)$$

and  $f_2(x_1, x_2) = ax_1(n) + bx_2(n) + c$ .

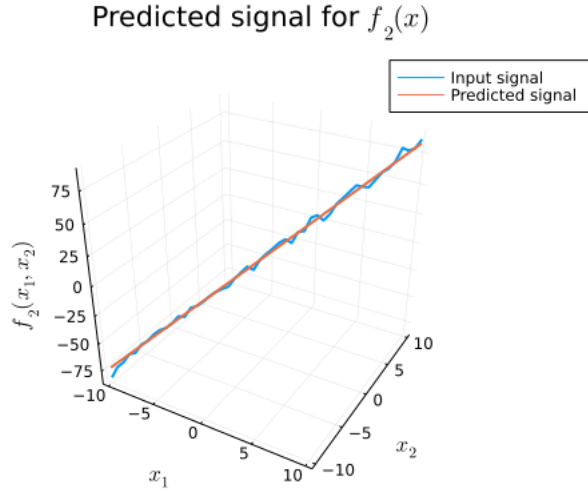
Table 2: ADALINE performance for regression problem

$f(\cdot)$	MSE mean	MSE standard deviation	RMSE mean	RMSE standard deviation
$5x(n) + 8$	9.69	2.84	3.07	0.47
$5x_1(n) + 3x_2(n) + 6$	9.93	4.27	3.08	0.66

Naturally, both curves could be properly estimated since they are linear functions. The Figure 5 shows the regression for the ADALINE model.



(a) ADALINE regression for  $5x + 7$



(b) ADALINE regression for  $5x_1(n) + 3x_2(n) + 6$

Figure 5: ADALINE regression.

### 3 Single Layer Perceptron

Although Rosenblatt's perceptron and ADALINE model can solve linear problems, it has only one output variable. A more sensible model for a multivariate class problem is a single-layer perceptron (SLP) consisting of  $J$  neurons, where each neuron receives the same input signal,  $\mathbf{x}(n)$ .

The matrix of all coefficients is given by

$$\mathbf{W}(n) = \begin{bmatrix} \mathbf{w}_1(n) & \mathbf{w}_2(n) & \cdots & \mathbf{w}_J(n) \end{bmatrix}^T \in \mathbb{R}^{J \times (N_a + 1)}, \quad (11)$$

where  $J$  is the number of classes (one neuron for each class),  $N_a$  is the number of attributes, and

$$\mathbf{w}_j(n) = \begin{bmatrix} w_{j0}(n) & w_{j1}(n) & \cdots & w_{jN_a}(n) \end{bmatrix}^T \in \mathbb{R}^{N_a + 1} \quad (12)$$

is the synaptic weight vector of the  $j$ th neuron, being  $w_{jk}(n)$  its  $k$ th weight.

The learning algorithm is given by

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \eta \boldsymbol{\delta}(n) \mathbf{x}^\top(n), \quad (13)$$

where

$$\mathbf{x}(n) = \begin{bmatrix} -1 & x_1(n) & x_2(n) & x_{N_a}(n) \end{bmatrix}^\top \in \mathbb{R}^{N_a+1} \quad (14)$$

is the input vector (including the bias), and

$$\boldsymbol{\delta}(n) = \begin{bmatrix} \delta_1(n) & \delta_2(n) & \delta_J(n) \end{bmatrix}^\top \in \mathbb{R}^J \quad (15)$$

is the vector of the local gradients, being  $\delta_j(n)$  the local gradient of the  $j$ th perceptron.

Let us define the vector of all induced local fields at the instant  $n$  as

$$\mathbf{v}(n) = \begin{bmatrix} v_1(n) & v_2(n) & v_J(n) \end{bmatrix}^\top = \mathbf{W}(n) \mathbf{x}(n). \quad (16)$$

Notice that

$$\begin{aligned} \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} &= \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \\ &= -\delta_j(n) x_i(n), \end{aligned} \quad (17)$$

where

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = x_i(n) \quad (18)$$

is the  $i$ th input at the instant  $n$ , and

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = -\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial x_i(n)} \frac{\partial x_i(n)}{\partial v_j(n)} \quad (19)$$

is the local gradient of the  $j$ th neuron, where  $e_j(n)$  error signal of the  $j$ th neuron at the instant  $n$ .

Note also that

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n), \quad (20)$$

$$\frac{\partial e_j(n)}{\partial x_i(n)} = -1, \quad (21)$$

and

$$\frac{\partial x_i(n)}{\partial v_j(n)} \triangleq \varphi'(v_j(n)). \quad (22)$$

By substituting the equations (20), (21), and (22) into (19), we get

$$\delta_j(n) = e_j(n)\varphi'(v_j(n)), \quad (23)$$

or in matricial notation,

$$\boldsymbol{\delta}(n) = \mathbf{e}(n) \odot \boldsymbol{\varphi}'(\mathbf{v}(n)), \quad (24)$$

where  $\odot$  is the Hadamard product and

$$\boldsymbol{\varphi}'(\mathbf{v}(n)) = \begin{bmatrix} \varphi'(v_1(n)) & \varphi'(v_2(n)) & \cdots & \varphi'(v_J(n)) \end{bmatrix}^\top \in \mathbb{R}^J. \quad (25)$$

The synaptic weights update is given by

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}. \quad (26)$$

Substituting the Equation (17) into (26), we have that

$$\Delta w_{ji}(n) = \eta \delta_j(n) x_i(n), \quad (27)$$

or in matricial notation

$$\Delta \mathbf{W}(n) = \eta \boldsymbol{\delta}(n) \mathbf{x}^\top(n). \quad (28)$$

Finally, the update equation is given by

$$\begin{aligned} \mathbf{W}(n+1) &= \mathbf{W}(n) + \Delta \mathbf{W}(n) \\ &= \mathbf{W}(n) + \eta \boldsymbol{\delta}(n) \mathbf{x}^\top(n). \end{aligned} \quad (29)$$

The Algorithm 2 summarizes the procedure of the SLP algorithm.

For the step function (MacCulloch and Pitts' activation function), its derivative does not exist, and the local gradient of the  $j$ th neuron is simply  $\delta_j(n) = e_j(n)$ . For this classification problem, the labels were encoded using the one-hot method.

The Figure 6 shows the heatmap for a dummy dataset consisting of  $K = 3$  classes,  $N_a = 2$  attributes, and  $N = 150$  instances. The classifier used the MacCulloch and Pitts' activation function and achieved a mean accuracy of 99.49% and a standard deviation of 0.0218. The SLP is also used for the iris dataset ( $K = 3$  classe,  $N_a = 4$  attributes,



---

**Algorithm 2:** Single-layer perceptron

---

```
Input:  $\mathbf{X}, \mathbf{D}$  // attributes and labels dataset
1 forall  $\{1, 2, \dots, N_r\}$  do
2    $\mathbf{W}(n) \leftarrow \text{initialize}$ 
3    $\mathbf{X}, \mathbf{D} \leftarrow \text{shuffle}$ 
4    $\mathbf{X}, \mathbf{D} \leftarrow \text{standardize}$ 
5    $(\mathbf{X}_{trn}, \mathbf{D}_{trn}), (\mathbf{X}_{tst}, \mathbf{D}_{tst}) \leftarrow \text{hold-out}$  // training and test dataset
6   forall  $\{1, 2, \dots, N_e\}$  do
7     forall Instances in the training dataset do
8        $\mathbf{v}(n) \leftarrow \mathbf{W}(n)\mathbf{x}(n)$ 
9        $\mathbf{y}(n) \leftarrow \varphi(\mathbf{v}(n))$ 
10       $\mathbf{e}(n) \leftarrow \mathbf{d}(n) - \mathbf{y}(n)$ 
11       $\boldsymbol{\delta}(n) \leftarrow \mathbf{e}(n) \odot \varphi'(\mathbf{v}(n))$ 
12       $\mathbf{W}(n+1) \leftarrow \mathbf{W}(n) + \eta \boldsymbol{\delta}(n) \mathbf{x}^T(n)$ 
13     $\mathbf{X}_{trn}, \mathbf{D}_{trn} \leftarrow \text{shuffle}$ 
14     $\text{accuracy} \leftarrow \text{test}(\mathbf{X}_{tst}, \mathbf{D}_{tst})$ 
15 Compute the mean and the standard deviation of all accuracies.
```

---

and  $N = 150$  attributes) and the column dataset ( $K = 3$  classes,  $N_a = 6$  attributes, and  $N = 310$  instances). For the iris dataset, the classifier achieved a mean accuracy of 88% with a standard deviation of 0.14, while for the column dataset the model achieved a mean accuracy of 77.66% with a standard deviation of 0.06.

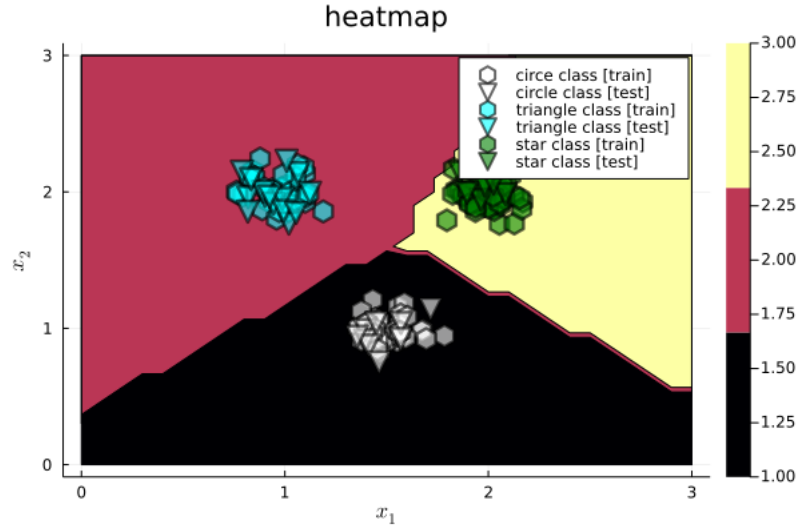


Figure 6: Heatmap of the dummy dataset.

Using the logistic function, the model achieved a mean accuracy of 100% for the dummy data. The heatmap for this dataset is shown in Figure 7.

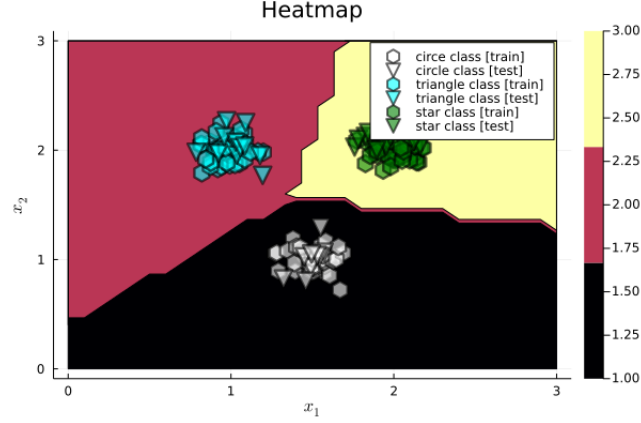
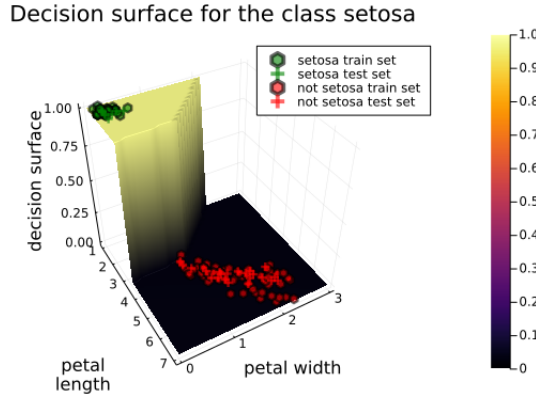
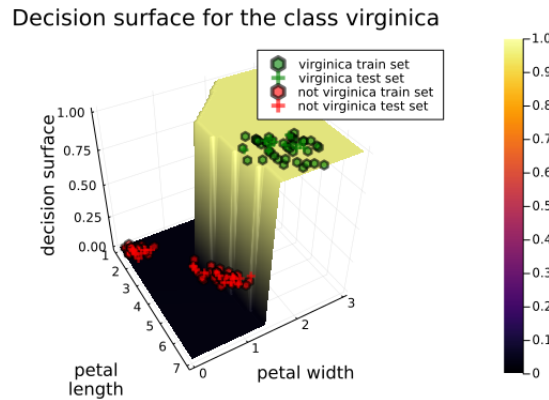


Figure 7: The dummy dataset for the SLP with logistic activation function.

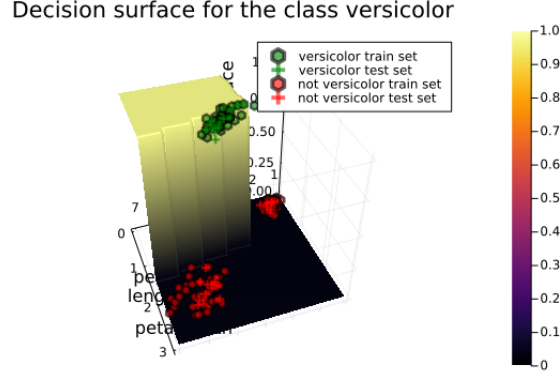
For the iris dataset, the SLP with logistic function reached a mean and a standard deviation of 87% and 0.16, respectively. The surface of decision for each class of iris data is shown in Figure 8. It is possible to notice that the classifier can solve the problem for the setosa class as it is linearly separable from the other classes for the attributes considered (petal length and petal width).



(a) Setosa class.



(b) Verginica class.



(c) Verginica class.

Figure 8: The iris dataset for the SLP with logistic activation function.

## 4 Multilayer Perceptron

Rosenblatt's perceptron and the single-layer perceptron use the LMS algorithm in the learning phase and are capable of estimating the gradient vector, which yields reasonable results for many problems. However, both architectures are restricted to the classification of linearly separable patterns.

To overcome the limitations of the aforementioned solutions, we introduce another neural network architecture, which employs  $L$  layers. This model is usually called Multilayer Perceptron (MLP). Each neuron uses a nonlinear<sup>6</sup> activation function that is differentiable, and the final output is able to solve nonlinear problems.

The presence of hidden layers makes the learning phase more complicated to devise since one must decide how the signal error at the output layer should propagate toward the input layer. A popular learning method used for MLP is the backpropagation algorithm, which in turn, is rooted in the LMS algorithm.

The backpropagation algorithm entails two phases:

- The *forward phase*: at the instant  $n$ , the synaptic weights of the network are fixed and the input signal,  $\mathbf{x}(n)$ , is propagated from the input to the output layer. At each neuron, the induced local field is computed and the output of the activation function is delivered to each neuron located on the layer at its right.
- The *backward phase*: signal error is produced at the output layer, where their weights are readily updated with the same procedure used in the SLP. Then, the synaptic weights of the hidden layers are updated, from the outmost hidden layer to the input layer, using the local gradients and the synaptic weights of the neurons on the layer at its right, in addition to the activation function derivative of the own neuron.

<sup>6</sup>If activation function were linear, the final result would be equal to the SLP output since  $\mathbf{y}(n)$  would simply be in a linear combination of  $\mathbf{x}(n)$ .

Since the update equation for the output layer was already derived in the SLP<sup>7</sup>, we will focus on finding the update equation of the  $l$ th hidden layer. Furthermore, we will considerate a dense neural network structure, that is, all neurons on the layer  $l + 1$  are connected through synaptic weights with all neurons on the layer  $l$ , for  $1 \leq l < L$ .

Beginning with the outmost hidden layer and recalling that the local gradient of the  $j$ th neuron on this layer is given by

$$\begin{aligned}\delta_j^{(L-1)}(n) &= -\frac{\partial \mathcal{E}(n)}{\partial v_j^{(L-1)}(n)} \\ &= -\frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} \frac{\partial y_j^{(L-1)}(n)}{\partial v_j^{(L-1)}(n)} \\ &= -\frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} \varphi'(v_j^{(L-1)}(n)),\end{aligned}\tag{30}$$

where the third equation follows that

$$\varphi'(v_j^{(L-1)}(n)) = \frac{\partial y_j^{(L-1)}(n)}{\partial v_j^{(L-1)}(n)}.\tag{31}$$

In these equations,  $v_j^{(l)}(n)$ ,  $y_j^{(l)}(n)$ , and  $\delta_j^{(l)}(n)$  are the induced local field, the output signal, and the local gradient of  $j$ th neuron on the  $l$ th layer, being  $y_0^{(l)}(n) \triangleq -1$  the input bias for the neurons on the layer  $l + 1$ .

We can derive the learning equation for the layer  $L - 1$  and generalize it to all hidden layers. Remember that  $y_j^{(L-1)}(n)$  is the  $j$ th input signal for the output layer ( $L$ th layer), and that

$$\frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} = \sum_{k=1}^{m_L} e_k(n) \frac{\partial e_k(n)}{\partial y_j^{(L-1)}(n)},\tag{32}$$

where  $m_L$  is the number of neurons on the  $L$ th layer. By using the chain rule, we have that

$$\frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} = \sum_{k=1}^{m_L} e_k(n) \frac{\partial e_k(n)}{\partial v_k^{(L)}(n)} \frac{\partial v_k^{(L)}(n)}{\partial y_j^{(L-1)}(n)},\tag{33}$$

but since  $e_k(n) = d_k(n) - \varphi(v_k^{(L)}(n))$ , it follows that

$$\frac{\partial e_k(n)}{\partial v_k^{(L)}(n)} = -\varphi'(v_k^{(L)}(n)).\tag{34}$$

---

<sup>7</sup>In the MLP, the unique difference is that the input signal is  $\mathbf{y}^{(L-1)}(n)$  instead of  $\mathbf{x}(n)$

Note that

$$v_k^{(L)}(n) = \sum_{j=0}^{m_L} w_{kj}^{(L)}(n) y_j^{(L-1)}(n). \quad (35)$$

Therefore,

$$\frac{\partial v_k^{(L)}(n)}{\partial y_j^{(L-1)}(n)} = w_{kj}^{(L)}(n) \quad (36)$$

Substituting the Equations (34) and (36) into (33), we get

$$\begin{aligned} \frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} &= - \sum_{k=1}^{m_L} e_k(n) \varphi'(v_k^{(L)}(n)) w_{kj}^{(L)}(n) \\ &= - \sum_{k=1}^{m_L} \delta_k^{(L)}(n) w_{kj}^{(L)}(n), \end{aligned} \quad (37)$$

where the second equation follows that  $\delta_k^{(L)} = e_k(n) \varphi'(v_k^{(L)}(n))$ . Finally, substituting Equation (37) into (30), the local gradient of the  $j$ th neuron on the  $(L-1)$ th hidden layer is given by

$$\delta_j^{(L-1)}(n) = \varphi'(v_j^{(L-1)}(n)) \sum_{k=1}^{m_L} \delta_k^{(L)}(n) w_{kj}^{(L)}(n), \quad (38)$$

or in matricial notation

$$\boldsymbol{\delta}^{(L-1)}(n) = \boldsymbol{\varphi}'(\mathbf{v}^{(L-1)}(n)) \odot \left( \tilde{\mathbf{W}}^{(L)\top}(n) \boldsymbol{\delta}^{(L)}(n) \right), \quad (39)$$

where

$$\tilde{\mathbf{W}}^{(L)}(n) = \begin{bmatrix} w_{11}^{(L)}(n) & w_{12}^{(L)}(n) & \cdots & w_{1m_{L-1}}^{(L)}(n) \\ w_{21}^{(L)}(n) & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ w_{m_L1}^{(L)}(n) & w_{m_L2}^{(L)}(n) & \cdots & w_{m_L m_{L-1}}^{(L)}(n) \end{bmatrix} \in \mathbb{R}^{m_L \times m_{L-1}} \quad (40)$$

is the synaptic weight matrix for the layer  $L$ , but without the weights of the bias.

The Equation (39) can be generalized to the  $l$ th layer:

$$\boldsymbol{\delta}^{(l)}(n) = \boldsymbol{\varphi}'(\mathbf{v}^{(l)}(n)) \odot \left( \tilde{\mathbf{W}}^{(l+1)\top}(n) \boldsymbol{\delta}^{(l+1)}(n) \right) \text{ for } 1 \leq l < L. \quad (41)$$

Following the same procedure for the SLP, we can derive the synaptic weights update,

which is given by

$$\Delta \mathbf{W}^{(l)}(n) = \eta \boldsymbol{\delta}^{(l)}(n) \mathbf{y}^{(l-1)\top}(n) \in \mathbb{R}^{m_l \times (m_{l-1}+1)}, \quad (42)$$

where

$$\mathbf{y}^{(l)}(n) = \begin{bmatrix} -1 & y_1^{(l)}(n) & y_2^{(l)}(n) & \cdots & y_{m_l}^{(l)}(n) \end{bmatrix}^\top \in \mathbb{R}^{m_l+1}, \quad (43)$$

$m_0$  is the length of the input vector without bias, and  $\mathbf{y}^{(0)}(n) \triangleq \mathbf{x}(n)$ . Therefore, the learning equation of the hidden layer is given by

$$\begin{aligned} \mathbf{W}^{(l)}(n+1) &= \mathbf{W}^{(l)}(n) + \Delta \mathbf{W}^{(l)}(n) \\ &= \mathbf{W}^{(l)}(n) + \eta \boldsymbol{\delta}^{(l)}(n) \mathbf{y}^{(l-1)\top}(n) \in \mathbb{R}^{m_l \times (m_{l-1}+1)}. \end{aligned} \quad (44)$$

The Algorithm 3 shows how the MLP uses the backpropagation algorithm.

---

**Algorithm 3:** Multilayer perceptron

---

```

Input:  $\mathbf{X}, \mathbf{D}$  // attributes and labels dataset
1 forall  $\{1, 2, \dots, N_r\}$  do
2    $\mathbf{W}(n) \leftarrow$  initialize
3    $\mathbf{X}, \mathbf{D} \leftarrow$  shuffle
4    $\mathbf{X}, \mathbf{D} \leftarrow$  standardize
5    $(\mathbf{X}_{trn}, \mathbf{D}_{trn}), (\mathbf{X}_{tst}, \mathbf{D}_{tst}) \leftarrow$  hold-out // training and test dataset
6   forall  $\{1, 2, \dots, N_e\}$  do
7     forall Instances in the training dataset do
8       // forward phase
9       for  $l \in \{1, 2, \dots, L\}$  do
10         $\mathbf{v}^{(l)}(n) \leftarrow \mathbf{W}^{(l)}(n) \mathbf{y}^{(l-1)}(n)$ 
11         $\mathbf{y}^{(l)}(n) \leftarrow \boldsymbol{\varphi}(\mathbf{v}^{(l)}(n))$ 
12      // backward phase
13       $\mathbf{e}(n) \leftarrow \mathbf{d}(n) - \mathbf{y}^{(L)}(n)$  // output layer
14       $\boldsymbol{\delta}^{(L)}(n) \leftarrow \boldsymbol{\varphi}'(\mathbf{v}^{(L)}(n)) \odot \mathbf{e}(n)$ 
15       $\mathbf{W}^{(L)}(n+1) \leftarrow \mathbf{W}^{(L)}(n) + \eta \boldsymbol{\delta}^{(L)}(n) \mathbf{y}^{(L-1)\top}(n)$ 
16      for  $l \in \{L-1, L-2, \dots, 1\}$  do // hidden layers
17         $\boldsymbol{\delta}^{(l)}(n) \leftarrow \boldsymbol{\varphi}'(\mathbf{v}^{(l)}(n)) \odot \left( \tilde{\mathbf{W}}^{(l+1)\top}(n) \boldsymbol{\delta}^{(l+1)}(n) \right)$ 
18         $\mathbf{W}^{(l)}(n+1) \leftarrow \mathbf{W}^{(l)}(n) + \eta \boldsymbol{\delta}^{(l)}(n) \mathbf{y}^{(l-1)\top}(n).$ 
19       $\mathbf{X}_{trn}, \mathbf{D}_{trn} \leftarrow$  shuffle
20     $accuracy \leftarrow$  test( $\mathbf{X}_{tst}, \mathbf{D}_{tst}$ )
21  Compute the mean and the standard deviation of all accuracies.
```

---

## 4.1 Grid search with $k$ -fold cross validation

For this homework, it is used an MLP with 2 layers, where the number of neurons in the hidden layers and the activation function (logistic function or hyperbolic tangent) are used in the grid search with  $k$ -fold cross-validation. The value of  $k$  varies with the size of the dataset, while the number of neurons on the output layer is always equal to 1 for regression problems or to the number of classes for classification problems<sup>8</sup>.

At each realization, before starting the training phase, the training dataset is utilized in the grid search with  $k$ -fold cross-validation. For each split,  $k-1$  folds are used iteratively for  $N_e$  epochs for validating, while the  $k$ th fold is reserved for test dataset. The accuracy obtained in the test dataset is stored and the process repeats for a different validation and test split. At the end of all combinations of test and validation splits, the mean accuracy is computed for that set of hyperparameters. The best set, that is, the one that obtains the higher mean accuracy, is used as the selected model. Note that the hyperparameters of the select model might vary per realization. The Algorithm 4 shows how the method of model selection works.

---

**Algorithm 4:** Grid search with  $k$ -fold cross validation

---

```
Input:  $\mathbf{X}, K$  //  $\mathbf{X}$  does not have the test instances
1  $\mathbf{X} \leftarrow$  shuffle
2 forall Set of hyperparameters do
3   for  $k \in \{1, 2, \dots, K\}$  do
4      $\mathbf{W}^{(l)}(n) \leftarrow$  initialize
5      $\mathbf{X}_{val}, \mathbf{X}_{tst} \leftarrow$  select the  $k$ th division of validation and test dataset.
6     forall  $\{1, 2, \dots, N_e\}$  do // for each epoch
7        $\mathbf{X}_{val} \leftarrow$  shuffle
8        $\mathbf{W}^{(l)}(n) \leftarrow$  validate using  $\mathbf{X}_{val}$  for a given set of hyperparameters
9     Save the accuracy for the trained  $\mathbf{W}^{(l)}(n)$  for the dataset  $\mathbf{X}_{tst}$ 
10  Save the mean accuracy for this set of hyperparameters
11 return The set of hyperparameters with the highest mean accuracy
```

---

## 4.2 Classification problem

For the MLP described in the Algorithm 3, the following datasets are analyzed for the classification problem:

- Iris:  $N_a = 4$  attributes,  $N = 150$  instances,  $K = 3$  classes.
- Vertebral column:  $N_a = 6$  attributes,  $N = 310$  instances,  $K = 3$  classes.

---

<sup>8</sup>When the number of classes is equal to 2 (as in the XOR problem), it is used only one neuron instead since it is sufficient to classify the problem.

- Dermatology:  $N_a = 33$  attributes,  $N = 366$  instances,  $K = 6$  classes.
- Breast Cancer Wisconsin:  $N_a = 10$  attributes,  $N = 699$  instances,  $K = 2$  classes.
- XOR problem:  $N_a = 2$  attributes,  $N = 200$  instances,  $K = 2$  classes.

The datasets with  $K = 2$  classes use only one neuron at its output, while datasets with  $K > 2$  use  $K$  neurons at the output layer.

#### 4.2.1 XOR dataset

For this classification problem, it is generated a dataset with 200 instances of the XOR problem. The equation

$$y(n) = x_1(n) \oplus x_2(n), \quad (45)$$

where the symbol  $\oplus$  indicates the exclusive-OR Boolean operator, generates a nonseparable surface problem that Rosenblatt's Perceptron and the single-layer Perceptron cannot solve. This problem is a specific case for a broad category of classification pattern called *unit hypercube*.

Since this problem has only two classes (0 or 1), it is used only one neuron at the output layer. With the trained MLP, the problem could be solved with 100% of mean accuracy. The select models use a hidden layer with 2 or 3 neurons and the select activation function was the logistic function. The Figure 9 shows the heatmap for the XOR problem.

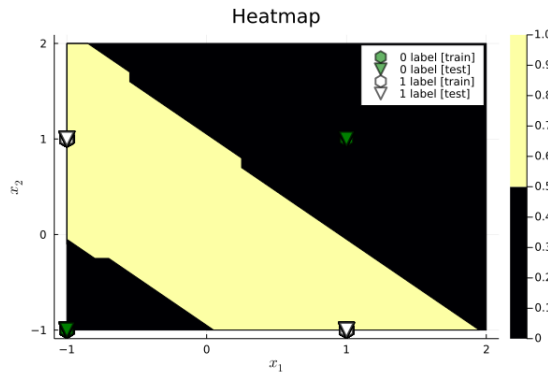


Figure 9: Heatmap for the XOR problem.

The Figure 10 shows the heatmap (or decision surface in a 3-D perspective) for another realization. It is interesting to note that the MLP can generalize differently and still correctly for the same dataset.



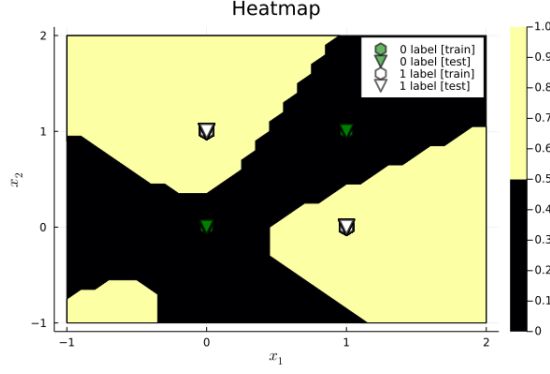


Figure 10: Heatmap for the XOR problem.

The confusion matrix is shown in Figure 11.

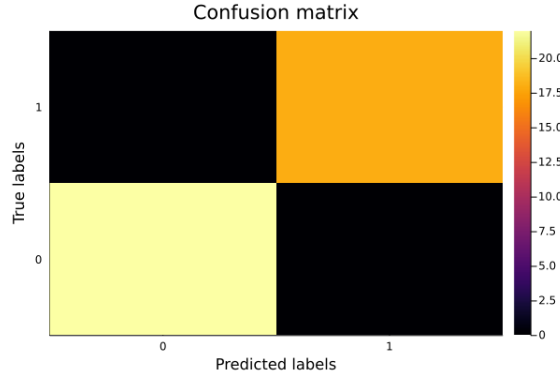
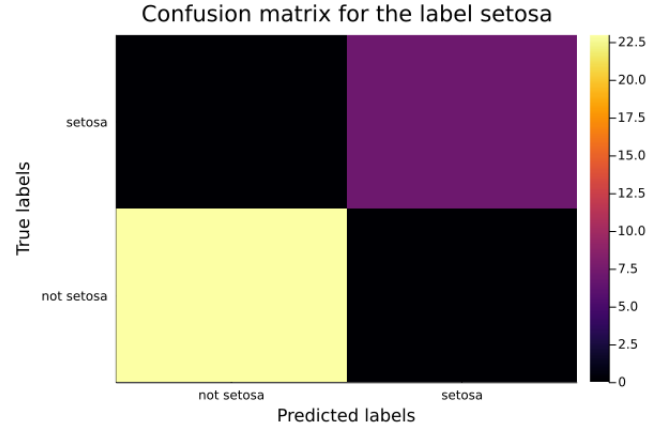


Figure 11: Confusion matrix for the XOR problem.

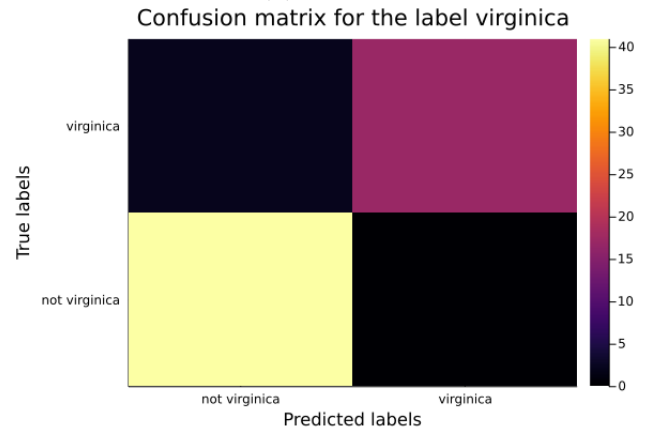
#### 4.2.2 Iris dataset

The iris dataset is used for an MLP with the same parameters as the XOR problem. However, the output layer now has 3 neurons since we have  $K = 3$  classes: setosa, virginica, and versicolor.

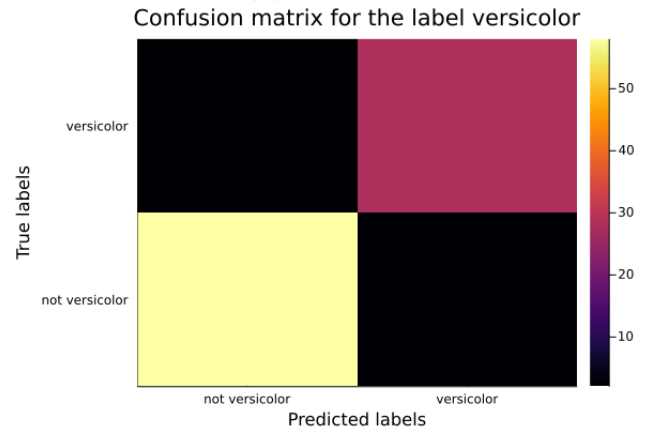
The number of neurons on the hidden layer varies around  $m_1 = 5$  and the activation function tends to the hyperbolic function for most realizations. The mean accuracy reached a value of 95.83% and a standard deviation of 0.031. These values overtake the performance achieved by the SLP architecture. The Figure 12 shows the confusion matrix for the realization with highest accuracy. One can observe that the MLP could recognize all patterns from the test dataset.



(a) Setosa.



(b) Virginica



(c) Versicolor

Figure 12: Confusion matrix for all classes of the Iris dataset.

## Column dataset

The cancer dataset was applied to the column dataset. The MLP reached a mean accuracy of 86.66%, with a standard deviation of 0.042. The Figure 13 shows the confusion matrix for each class.

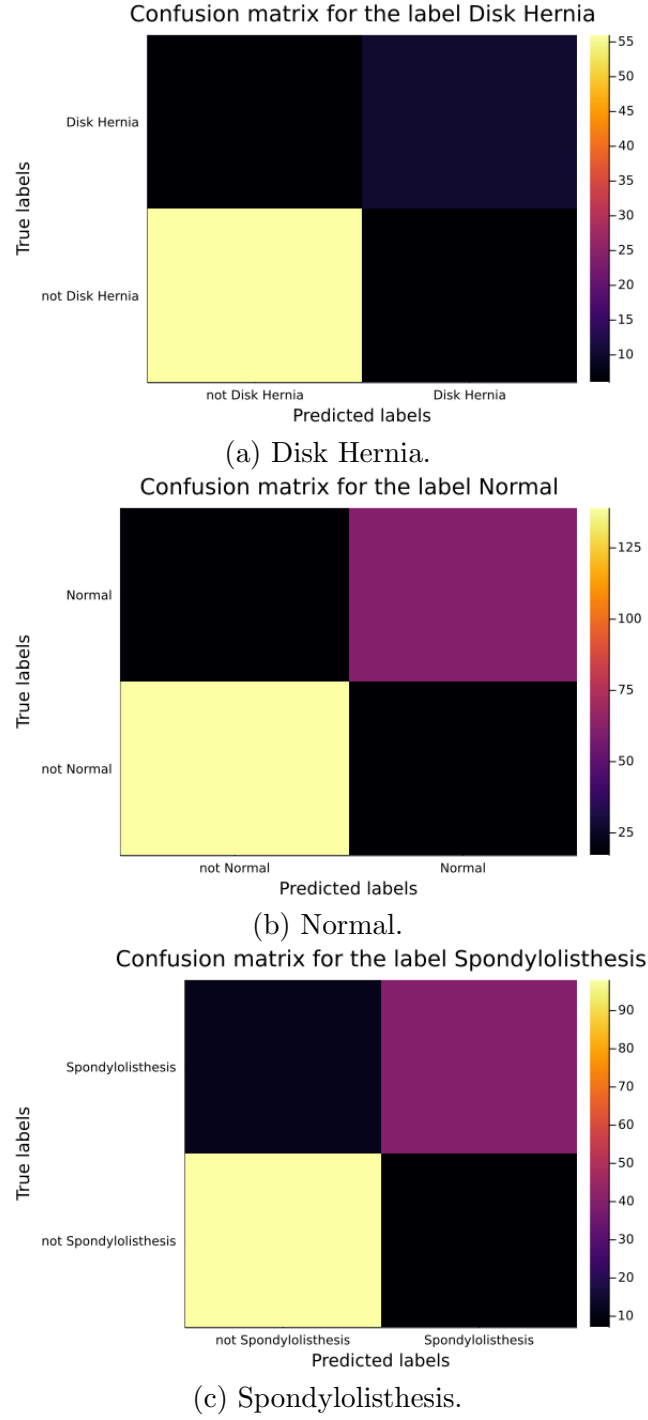


Figure 13: Confusion matrix for all classes of the column dataset.

#### 4.2.3 Breast cancer dataset

For the breast cancer dataset, the MLP reached a mean accuracy of 96.44% with a standard deviation of 0.0118. The Figure 14 show the confusion matrix for this case.

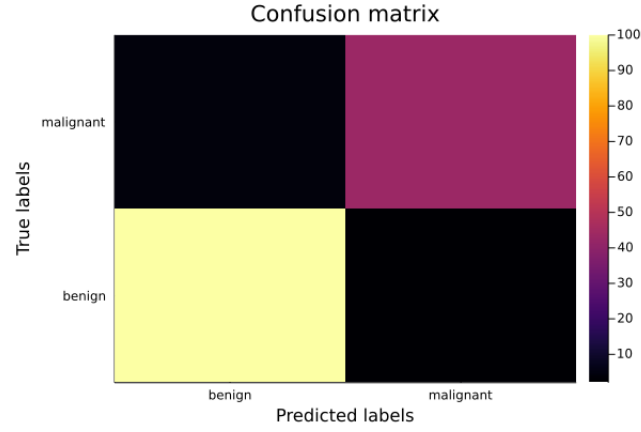
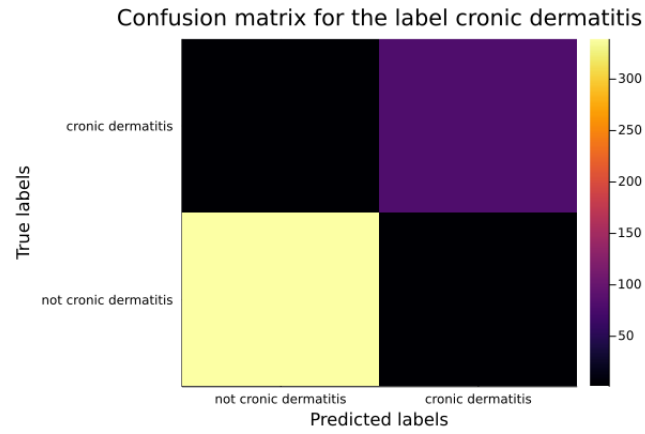


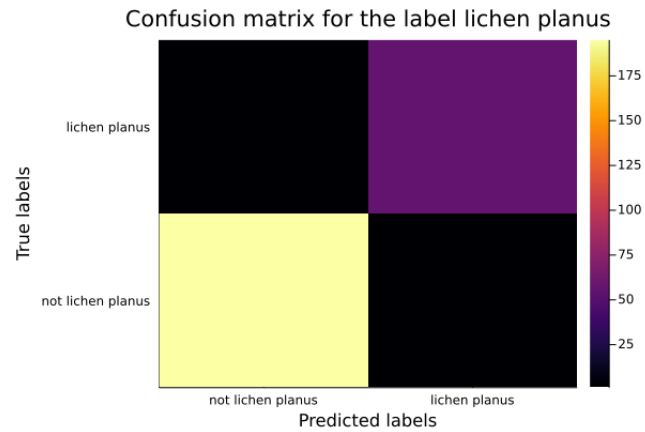
Figure 14: Confusion matrix for the cancer dataset.

#### 4.2.4 Dermatology

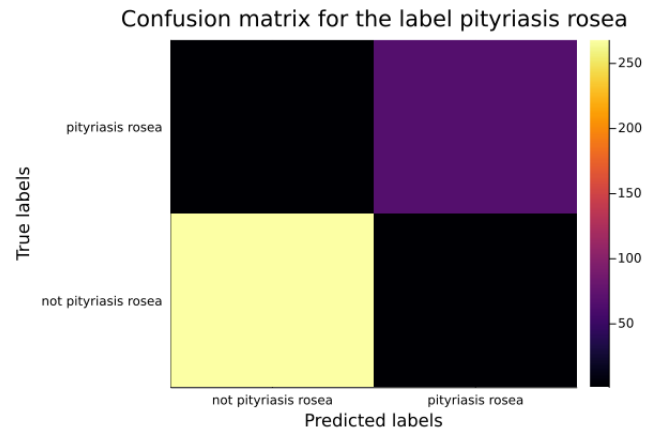
For the dermatology dataset, the MLP reached an accuracy of 97.15% with a standard deviation of 0.01836. The Figure 14 show the confusion matrix for this case.



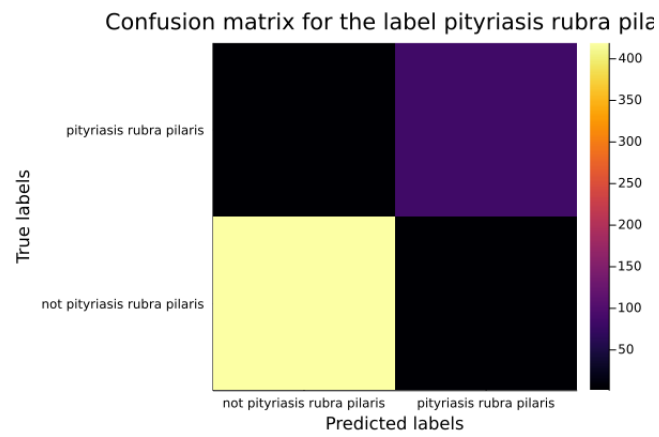
(a) Cronic dermatitis.



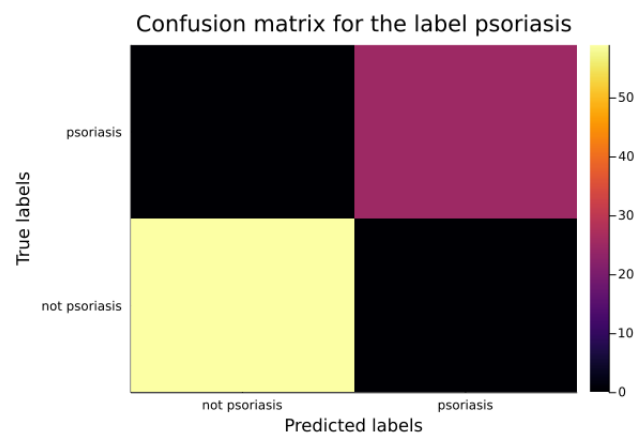
(b) Lichen planus.



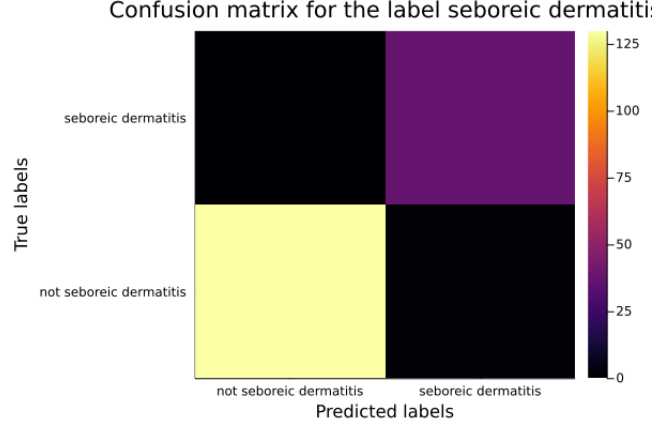
(c) Pityriasis rosea.



(d) Pityriasis rubra pilaris.



(e) Psoriasis.



(f) Seboreic dermatitis.

Figure 15: Confusion matrix for all classes of the dermatology dataset.

## Regression problem

For the regression task, the following datasets are considered

- Dummy dataset: Artificial dataset where the desired signal is  $d(n) = 3 \sin(x(n)) + 1$ .
- Electric Motor Temperature: The usage of a set of parameters of an electric motor (current and voltages in d/q coordinates, motor speed, torque, etc...) to estimate its temperature. In total, the neural network tries to estimate the electric motor temperature in five areas: stator tooth, stator winding, stator yoke, permanent magnet, and coolant.
- Car Fuel Consumption: An attempt to estimate the car consumption based on a series of parameters, such as the gasoline type, the weather condition, the usage of the air conditioner, etc...

All datasets were normalized to the range  $[-1, 1]$ . This makes the neural network output signal range suitable for the normalized desired signal when it is employed the hyperbolic tangent as the activation function. Moreover, the output layer has only one neuron since it is a regression problem. The unique exception is the Electric Motor Temperature dataset, in which the MLP estimates five features using five neurons on the output layer.

The Figure 17 shows the regression for the sine function and the Table 3 summarizes the results in terms of the mean RMSE and its standard deviation obtained for each dataset.

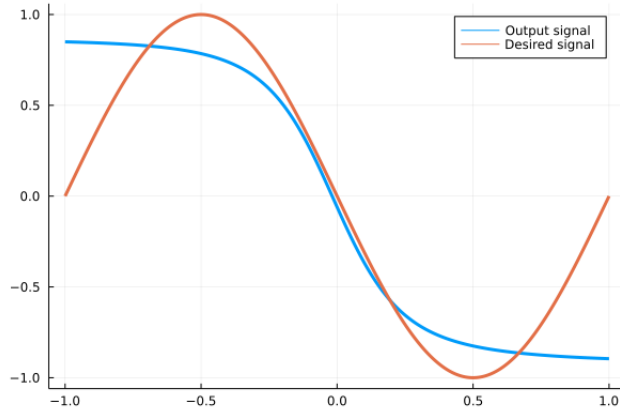


Figure 16: Regression solution for the sine function problem.

Table 3: RMSE for several datasets of regression problem

Dataset	mean RMSE	standard deviation
Car fuel	0.1417	0.022
Electric Motor Temperature - Coolant	0.3295	0.08
Electric Motor Temperature - Stator winding	0.0337	0.3378
Electric Motor Temperature - Stator tooth	0.033	0.3378
Electric Motor Temperature - Permanent magnet	0.0512	0.3355
Electric Motor Temperature - Stator yoke	0.03981	0.3371

As a regression example, the Figure shows the estimated stator winding temperature for one session<sup>9</sup>. It is possible to see that the trained neural network can successively estimate the stator winding temperature.

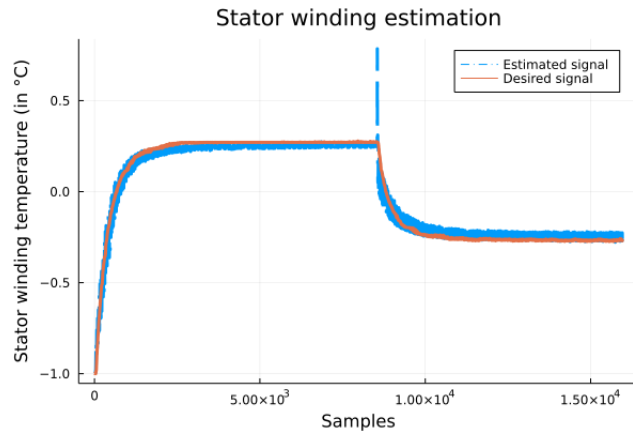


Figure 17: Regression solution for the sine function problem.

<sup>9</sup>The data set is divided into several measurement sessions, in which one session consists of a six-hour test bench.