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 01 - Rosenblatt's perceptron

This work considers a classification problem for a multivariate dataset. The Rosenblatt's perceptron is utilized to classify the iris flower dataset. The problem consists in classifying one class among four subspecies (Setosa, Virginica, and Versicolor).

The Rosenblatt's perceptron comprises a neuron mathematical model, introduced by McCulloch and Pitts in 1943, with 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 classification 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

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

where

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

and

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

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 $x_0(n) \triangleq +1$[1] are, respectively, the bias and its input.

The machine learning algorithms settle on the well-established theory of adaptive filters. Particularly for the Rosenblatt's perceptron, it is utilized the Least-Mean-Square (LMS) algorithm, which aims to make an instantaneous approximation of the gradient

[1] Depending on the author, it can the defined as -1.

vector. The optimization algorithm is given by

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta\hat{\mathbf{g}}(n), \tag{4}$$

where η is the step-learning hyperparameter and $\hat{\mathbf{g}}(n) \triangleq \nabla\mathscr{E}(\mathbf{w})$ is the stochastic approximation of the gradient vector, being $\mathscr{E}(\mathbf{w})$ the cost function and ∇ the vector differential operator. The Equation (1) passes through the step function, $\varphi(\cdot)$, generating the perceptron output, $y(n) = \varphi(v(n)) \in \{0, 1\}$. This signal is compared to the desired value, $d(n) \in \{0, 1\}$, and produces the error signal, $e(n) = d(n) - y(n) \in \{-1, 0, 1\}$, which indicates whether the perceptron misclassified or not.

The LMS algorithm uses the instantaneous value of the MSE (Mean-Squared Error) cost function, that is,

$$\mathscr{E}(\mathbf{w}) = \frac{1}{2}e^2(n). \tag{5}$$

Differentiating this equation with respect to the synaptic weights, we get

$$\hat{\mathbf{g}}(n) = \frac{\partial\mathscr{E}(\mathbf{w})}{\partial\mathbf{w}(n)} = -\mathbf{x}(n)e(n). \tag{6}$$

Substituting (6) into (4), it yields the learning equation, given by

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\mathbf{x}(n)e(n). \tag{7}$$

The Algorithm 1 summarizes the procedure utilized for the Rosenblatt's perceptron, including data preparation techniques, such as hand-out and data shuffling. The method utilizes $N_r = 20$ independent realizations, and passes through the training set $N_e = 100$ epochs. At the end of each realization, it is stored the accuracy[2] reached by the test data, and the accuracy of all realizations are 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 $K = 3$ classes (Setosa, Versicolour, and Virginica). It was chosen a ratio of $80\% - 20\%$ for the training and test datasets, respectively [3].

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[4].

[2]Accuracy is defined as the ratio of the number of correct predictions by the total number of predictions

[3]The values of N_e, N_r, and the train-test ratio is maintained throughout this homework.

[4]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.

Algorithm 1: Rosenblatt's perceptron

Input: $\mathbf{X}, \mathbf{d}$ // attributes and labels dataset

forall $\{1, 2, \cdots, N_r\}$ **do**
- $\mathbf{w}(n) \leftarrow$ initialize
- $\mathbf{X}, \mathbf{d} \leftarrow$ shuffle
- $(\mathbf{X}_{trn}, \mathbf{d}_{trn}), (\mathbf{X}_{tst}, \mathbf{d}_{tst}) \leftarrow$ hold-out // training and test dataset
- **forall** $\{1, 2, \cdots, N_e\}$ **do**
  - **forall** *Instancies in the training dataset* **do**
    - $v(n) \leftarrow \mathbf{w}^{\mathsf{T}}(n)\,\mathbf{x}(n)$
    - $y(n) \leftarrow \varphi(v(n))$
    - $e(n) \leftarrow d(n) - y(n)$
    - $\mathbf{w}(n+1) \leftarrow \mathbf{w}(n) + \eta \mathbf{x}(n) e(n)$
  - $\mathbf{X}_{trn}, \mathbf{d}_{trn} \leftarrow$ shuffle
- $accuracy \leftarrow \text{test}(\mathbf{X}_{tst}, \mathbf{d}_{tst})$

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

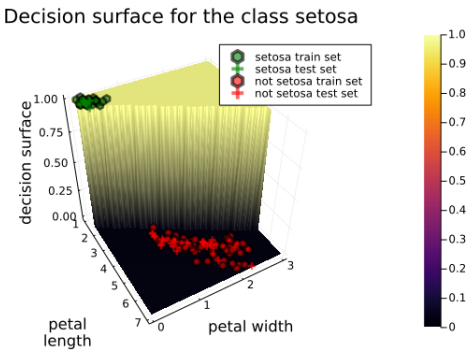


Figure 1: Decision surface of setosa class.

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.

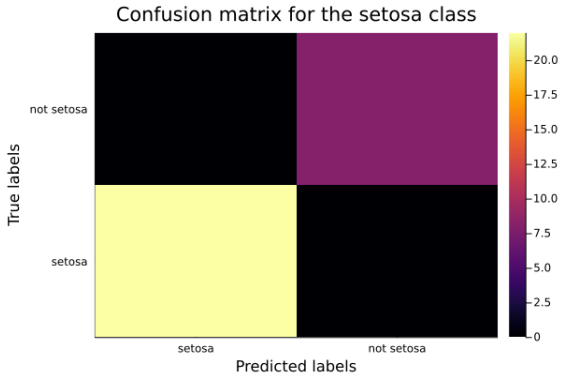


Figure 2: confusion matrix for setosa class.

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

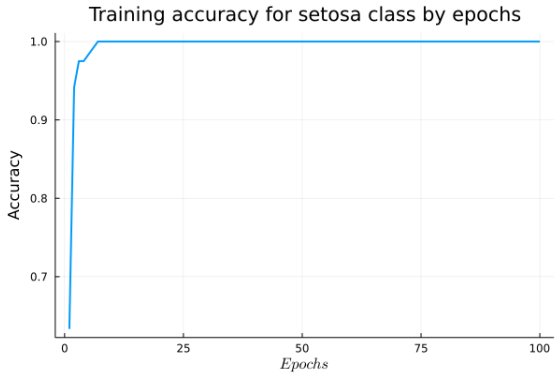


Figure 3: Training dataset evolution for the setosa classification.

For a dummy dataset with $K = 4$ classes, the Rosenblatt's perceptron achieved a mean accuracy of 97.5% and a standard deviation of 0.05. 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.

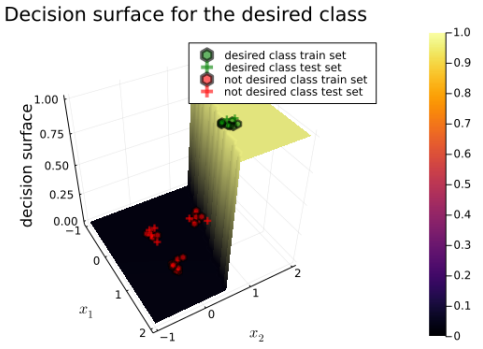


Figure 4: Decision surface for the desired class.

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 is toward 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

$$\begin{bmatrix} 1 & x(n) \end{bmatrix} \in \mathbb{R}^2 \tag{8}$$

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

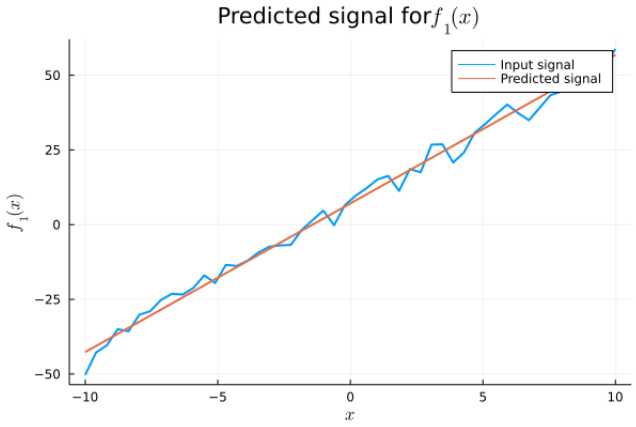
$$\begin{bmatrix} 1 & x_1(n) & x_2(n) \end{bmatrix} \in \mathbb{R}^3 \tag{9}$$

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

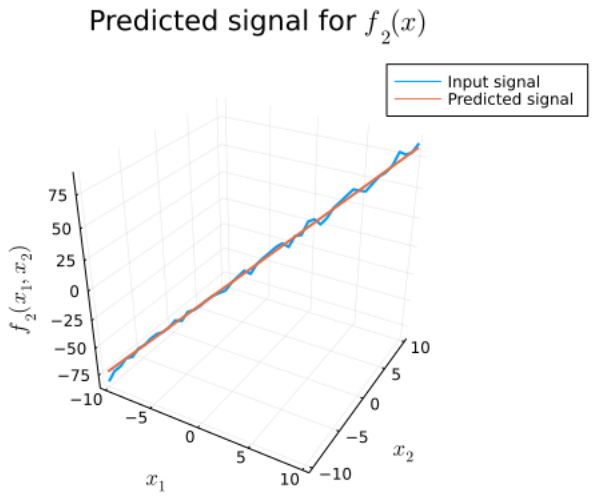
Table 2: ADALINE performance for regression problem

$f(\cdot)$	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 can solve linear problems, it has only one output variable. A more reasonable 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

$$\mathbf{W}(n) = \begin{bmatrix} \mathbf{w}_1(n)^\mathsf{T} & \mathbf{w}_2(n)^\mathsf{T} & \cdots & \mathbf{w}_J(n)^\mathsf{T} \end{bmatrix}^\mathsf{T} \in \mathbb{R}^{J\times(N_a+1)}, \tag{10}$$

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

and

$$\mathbf{w}_j(n) = \begin{bmatrix} w_{j0}(n) & w_{j1}(n) & \cdots & w_{jN_a}(n) \end{bmatrix}^\mathsf{T} \in \mathbb{R}^{N_a+1} \tag{11}$$

is the synaptic weight vector of the jth neuron, being $w_{jk}(n)$ its kth weight.

The learning algorithm is given by

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \eta \boldsymbol{\delta}(n)\mathbf{x}(n)^\mathsf{T}, \tag{12}$$

where

$$\mathbf{x}(n) = \begin{bmatrix} -1 & x_1(n) & x_2(n) & x_{N_a}(n) \end{bmatrix}^\mathsf{T} \in \mathbb{R}^{N_a+1} \tag{13}$$

is the input vector (including the bias), and

$$\boldsymbol{\delta}(n) = \begin{bmatrix} \delta_1(n) & \delta_2(n) & \delta_J(n) \end{bmatrix}^\mathsf{T} \in \mathbb{R}^J \tag{14}$$

is the vector of the local gradients, being $\delta_j(n)$ the local gradient of the jth perceptron. Let us define the vector of all induced local fields at the instant n as

$$\mathbf{v}(n) = \begin{bmatrix} v_1(n) & v_2(n) & v_J(n) \end{bmatrix}^\mathsf{T} = \mathbf{W}(n)\mathbf{x}(n). \tag{15}$$

Notice that

$$\begin{aligned} \frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} &= \frac{\partial \mathscr{E}(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \\ &= -\delta_j(n) x_i(n), \end{aligned} \tag{16}$$

where

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = x_i(n) \tag{17}$$

is the ith input at the instant n, and

$$\delta_j(n) = -\frac{\partial \mathscr{E}(n)}{\partial v_j(n)} = -\frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial x_i(n)} \frac{\partial x_i(n)}{\partial v_j(n)} \tag{18}$$

is the local gradient of the jth neuron. In this equation, $e_j(n)$ and $v_j(n)$ are the error and the induced local field of the neuron j, respectively, and $w_{ji}(n)$ is the ith synaptic weight for the jth neuron at the instant n.

Note also that

$$\frac{\partial \mathscr{E}(n)}{\partial e_j(n)} = e_j(n), \tag{19}$$

$$\frac{\partial e_j(n)}{\partial x_i(n)} = -1, \tag{20}$$

and

$$\frac{\partial x_i(n)}{\partial v_j(n)} \triangleq \varphi'(v_j(n)). \tag{21}$$

By substituting the equations (19), (20), and (21) into (18), we get

$$\delta_j(n) = e_j(n)\varphi'(v_j(n)), \tag{22}$$

or in matricial notation,

$$\boldsymbol{\delta}(n) = \mathbf{e}(n) \odot \boldsymbol{\varphi}'(\mathbf{v}(n)), \tag{23}$$

where $\odot$ is the Hadamard product and

$$\boldsymbol{\varphi}'(\mathbf{v}(n)) = \begin{bmatrix} \varphi'(v_1(n)) & \varphi'(v_2(n)) & \cdots & \varphi'(v_J(n)) \end{bmatrix}^{\mathsf{T}} \in \mathbb{R}^J. \tag{24}$$

The synaptic weights update is given by

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)}. \tag{25}$$

Substituting the Equation (16) into (25), we have that

$$\Delta w_{ji}(n) = \eta \delta_j(n) x_i(n), \tag{26}$$

or in matricial notation

$$\Delta \mathbf{W}(n) = \eta \boldsymbol{\delta}(n) \mathbf{x}(n)^{\mathsf{T}}. \tag{27}$$

Finally, the update function is given by

$$\begin{aligned} \mathbf{W}(n+1) &= \mathbf{W}(n) + \Delta \mathbf{W}(n) \\ &= \mathbf{W}(n) + \eta \boldsymbol{\delta}(n) \mathbf{x}(n)^{\mathsf{T}}. \end{aligned} \tag{28}$$

The Algorithm 2 summarizes the procedure of the SLP algorithm.

Algorithm 2: Single-layer perceptron

Input: $\mathbf{X}, \mathbf{D}$ // attributes and labels dataset

forall $\{1, 2, \cdots, N_r\}$ **do**
- $\mathbf{W}(n) \leftarrow$ initialize
- $\mathbf{X}, \mathbf{D} \leftarrow$ shuffle
- $(\mathbf{X}_{trn}, \mathbf{D}_{trn}), (\mathbf{X}_{tst}, \mathbf{D}_{tst}) \leftarrow$ hold-out // training and test dataset
- **forall** $\{1, 2, \cdots, N_e\}$ **do**
  - **forall** *Instancies in the training dataset* **do**
    - $\mathbf{v}(n) \leftarrow \mathbf{W}(n)\mathbf{x}(n)$
    - $\mathbf{y}(n) \leftarrow \boldsymbol{\varphi}(\mathbf{v}(n))$
    - $\mathbf{e}(n) \leftarrow \mathbf{d}(n) - \mathbf{y}(n)$
    - $\boldsymbol{\delta}(n) \leftarrow \mathbf{e}(n) \odot \boldsymbol{\varphi}'(\mathbf{v}(n))$
    - $\mathbf{W}(n+1) \leftarrow \mathbf{W}(n) + \eta\boldsymbol{\delta}(n)\mathbf{x}(n)^{\mathsf{T}}$
  - $\mathbf{X}_{trn}, \mathbf{D}_{trn} \leftarrow$ shuffle
- $accuracy \leftarrow \text{test}(\mathbf{X}_{tst}, \mathbf{D}_{tst})$

For the step function (MacCulloch and Pitts' activation function), its derivative does not exist, and the local gradient of the jth 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 same classifier was used for the iris dataset ($K = 3$ classe, $N_a = 4$ attributes, 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.

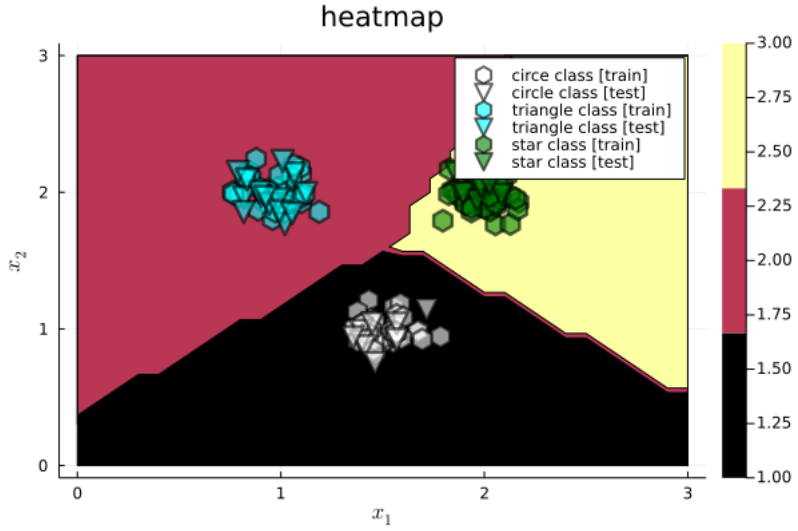


Figure 6: Heatmap of the dummy dataset.

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.

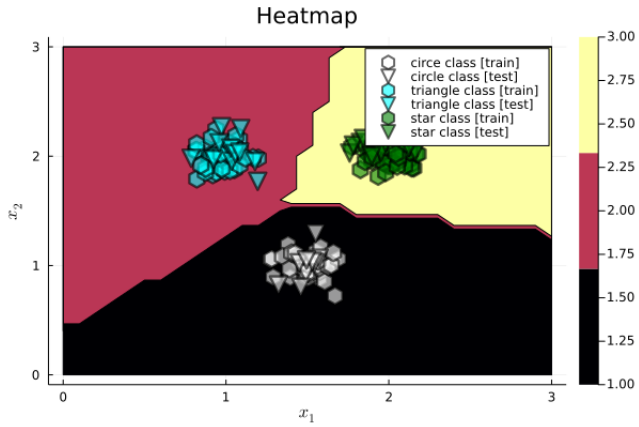
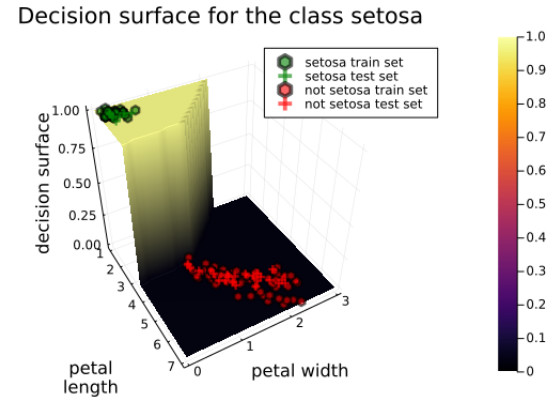
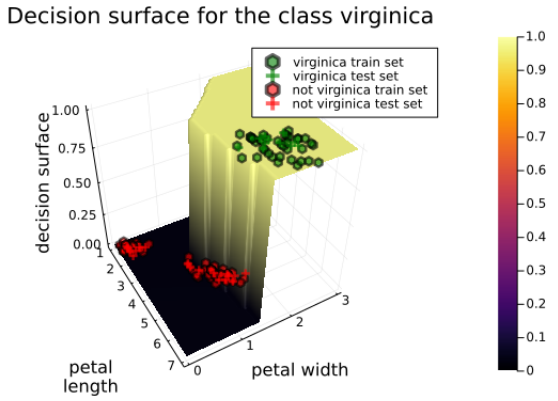


Figure 7: The dummy dataset for the SLP with logistic activation function.

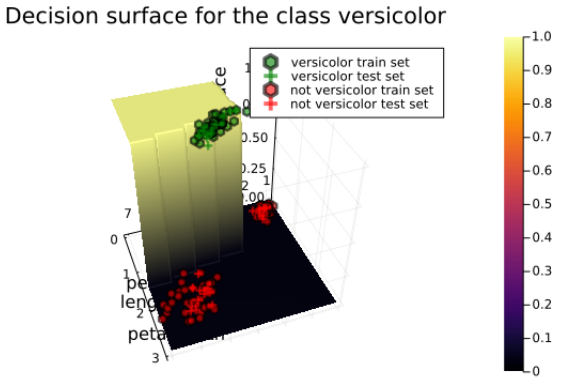
For the iris dataset, SLP with logistic function reached a mean and a standard deviation of 87% and 0.16, respectively. The surface of decision for each class of iris data is shown in Figure 8. It is possible to notice that the classifier can solve the problem for the setosa class as it is linearly separable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



(a) Setosa class



(b) Verginica class



(c) Versicolor 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 a new neural network architecture, which employs L layers. This model is usually called Multilayer Perceptron (MLP). Each neuron uses a nonlinear 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.

Since the update equation for the output layer was already derived in the SLP[5], we will focus on finding the update equation of the lth 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 jth neuron on this layer is given by

$$\begin{aligned}
\delta_j^{(L-1)}(n) &= -\frac{\partial \mathscr{E}(n)}{\partial v_j^{(L-1)}(n)} \\
&= -\frac{\partial \mathscr{E}(n)}{\partial y_j^{(L-1)}(n)} \frac{\partial y_j^{(L-1)}(n)}{\partial v_j^{(L-1)}(n)} \\
&= -\frac{\partial \mathscr{E}(n)}{\partial y_j^{(L-1)}(n)} \varphi'(v_j^{(L-1)}(n)),
\end{aligned} \tag{29}$$

where the third equation follows that

$$\varphi'(v_j^{(L-1)}(n)) = \frac{\partial y_j^{(L-1)}(n)}{\partial v_j^{(L-1)}(n)}. \tag{30}$$

[5]In the MLP, the unique difference is that the input signal is $\mathbf{y}^{(L-1)}(n)$ instead of $\mathbf{x}(n)$

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 jth neuron on the lth 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 jth input signal on the output layer (Lth layer), and that

$$\frac{\partial \mathscr{E}(n)}{\partial y_j^{(L-1)}(n)} = \sum_{k=1}^{m_L} e_k(n) \frac{\partial e_k(n)}{\partial y_j^{(L-1)}(n)}, \tag{31}$$

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

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

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

$$\frac{\partial e_k(n)}{\partial v_k^{(L)}(n)} = -\varphi(v_k^{(L)}(n)). \tag{33}$$

Note that

$$v_k^{(L)}(n) = \sum_{j=0}^{m_L} w_{kj}^{(L)}(n) y_j^{(L-1)}(n). \tag{34}$$

Therefore,

$$\frac{\partial v_k^{(L)}(n)}{\partial y_j^{(L-1)}(n)} = w_{kj}^{(L)}(n) \tag{35}$$

Substituting the Equations (33) and (35) into (32), we get

$$\begin{aligned} \frac{\partial \mathscr{E}(n)}{\partial y_j^{(L-1)}(n)} &= -\sum_{k=1}^{m_L} e_k(n) \varphi(v_k^{(L)}(n)) w_{kj}^{(L)}(n) \\ &= -\sum_{k=1}^{m_L} \delta_k^{(L)}(n) w_{kj}^{(L)}(n), \end{aligned} \tag{36}$$

where the second equation follows that $\delta_k^{(L)} = e_k(n)\varphi(v_k^{(L)}(n))$. Finally, substituting Equation (36) into (29), the local gradient of the jth neuron on the $(L-1)$th hidden layer

is given by

$$\delta_j^{(L-1)}(n) = \varphi'(v_j^{(L-1)}(n)) \sum_{k=1}^{m_L} \delta_k^{(L)}(n) w_{kj}^{(L)}(n), \tag{37}$$

or in matricial notation

$$\boldsymbol{\delta}^{(L-1)}(n) = \boldsymbol{\varphi}'(\mathbf{v}^{(L-1)}(n)) \odot \tilde{\mathbf{W}}^{(L)}(n)^{\mathsf{T}} \boldsymbol{\delta}^{(L)}(n), \tag{38}$$

where

$$\tilde{\mathbf{W}}^{(L)}(n) = \begin{bmatrix} w_{11}(n) & w_{12}(n) & \cdots & w_{1m_{L-1}}(n) \\ w_{21}(n) & \ddots & \cdots & \vdots \\ \vdots & \ddots & & \vdots \\ w_{m_L1}(n) & w_{m_L2}(n) & \cdots & w_{m_Lm_{L-1}}(n) \end{bmatrix} \in \mathbb{R}^{m_L \times m_{L-1}} \tag{39}$$

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

The Equation (38) can be generalized to the lth layer:

$$\boldsymbol{\delta}^{(l)}(n) = \boldsymbol{\varphi}'(\mathbf{v}^{(l)}(n)) \odot \tilde{\mathbf{W}}^{(l+1)}(n)^{\mathsf{T}} \boldsymbol{\delta}^{(l+1)}(n) \text{ for } 1 \leq l < L. \tag{40}$$

Following the same procedure for the SLP, we can derive the synaptic weights update, which is given by

$$\Delta \mathbf{W}^{(l)}(n) = \eta \boldsymbol{\delta}^{(l)}(n) \mathbf{y}^{(l-1)}(n)^{\mathsf{T}} \in \mathbb{R}^{m_l \times (m_{l-1}+1)}, \tag{41}$$

where

$$\mathbf{y}^{(l)}(n) = \begin{bmatrix} -1 & y_1^{(l)}(n) & y_2^{(l)}(n) & \cdots & y_{m_l}^{(l)}(n) \end{bmatrix}^{\mathsf{T}} \in \mathbb{R}^{m_l+1}, \tag{42}$$

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

$$\begin{aligned} \mathbf{W}^{(l)}(n+1) &= \mathbf{W}^{(l)}(n) + \Delta \mathbf{W}^{(l)}(n) \\ &= \mathbf{W}^{(l)}(n) + \eta \boldsymbol{\delta}^{(l)}(n) \mathbf{y}^{(l-1)}(n)^{\mathsf{T}} \in \mathbb{R}^{m_l \times (m_{l-1}+1)}. \end{aligned} \tag{43}$$

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

Algorithm 3: Multilayer perceptron

Input: $\mathbf{X}, \mathbf{D}$ // attributes and labels dataset

forall $\{1, 2, \cdots, N_r\}$ **do**
  $\mathbf{W}(n) \leftarrow$ initialize
  $\mathbf{X}, \mathbf{D} \leftarrow$ shuffle
  $(\mathbf{X}_{trn}, \mathbf{D}_{trn}), (\mathbf{X}_{tst}, \mathbf{D}_{tst}) \leftarrow$ hold-out // training and test dataset
  forall $\{1, 2, \cdots, N_e\}$ **do**
    forall *Instancies in the training dataset* **do**
      // forward phase
      for $l \in \{1, 2, ..., L\}$ **do**
        $\mathbf{v}^{(l)}(n) \leftarrow \mathbf{W}^{(l)}(n)\mathbf{y}^{(l-1)}(n)$
        $\mathbf{y}^{(l)}(n) \leftarrow \boldsymbol{\varphi}(\mathbf{v}^{(l)}(n))$
      // backward phase
      $\mathbf{e}(n) \leftarrow \mathbf{d}(n) - \mathbf{y}^{(L)}(n)$ // output layer
      $\boldsymbol{\delta}^{(L)}(n) \leftarrow \boldsymbol{\varphi}'(\mathbf{v}^{(L)}(n)) \odot \mathbf{e}(n)$
      $\mathbf{W}^{(L)}(n+1) \leftarrow \mathbf{W}^{(L)}(n) + \eta\boldsymbol{\delta}^{(L)}(n)\mathbf{y}^{(L-1)}(n)^{\mathsf{T}}$
      for $l \in \{L-1, L-2, ..., 1\}$ **do** // hidden layers
        $\boldsymbol{\delta}^{(l)}(n) \leftarrow \boldsymbol{\varphi}'(\mathbf{v}^{(l)}(n)) \odot \tilde{\mathbf{W}}^{(l+1)}(n)^{\mathsf{T}}\boldsymbol{\delta}^{(l+1)}(n)$
        $\mathbf{W}^{(l)}(n+1) \leftarrow \mathbf{W}^{(l)}(n) + \eta\boldsymbol{\delta}^{(l)}(n)\mathbf{y}^{(l-1)}(n)^{\mathsf{T}}.$
    $\mathbf{X}_{trn}, \mathbf{D}_{trn} \leftarrow$ shuffle
  $accuracy \leftarrow \text{test}(\mathbf{X}_{tst}, \mathbf{D}_{tst})$

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[6].

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 training, while the kth fold is reserved for validation dataset. The accuracy obtained in the validation dataset is stored and the process repeats for a different training and validation split. At the end of all combinations of validation and training 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. The Algorithm 4 shows how the method of model selection works.

[6]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.

Algorithm 4: Grid search with k-fold cross validation

Input: $\mathbf{X}, K$ `// X does not have the test instances`

$\mathbf{X} \leftarrow$ shuffle
forall *Set of hyperparameters* **do**
  for $k \in \{1, 2, ..., K\}$ **do**
    $\mathbf{W}(n) \leftarrow$ initialize
    $\mathbf{X}_{trn}, \mathbf{X}_{val} \leftarrow$ select the kth division of training and validation dataset.
    forall $\{1, 2, ..., N_e\}$ **do** `// for each epoch`
      $\mathbf{X}_{trn} \leftarrow$ shuffle
      $\mathbf{W}(n) \leftarrow$ train with a given set of hyperparameters
    Save the accuracy for the trained $\mathbf{W}(n)$ for the dataset $\mathbf{X}_{val}$
  Save the mean accuracy for this set of hyperparameters
return *The set of hyperparameters with the highest mean accuracy*

4.1 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.
- Dermatology: $N_a = 33$ attributes, $N = 366$ instances, $K =?$ 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, with the datasets with $K > 2$ use K neurons at the output layer.

4.1.1 XOR dataset

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

$$y(n) = x_1(n) \oplus x_2(n), \tag{44}$$

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*. This is a well-known nonlinear classification

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 Figure