



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

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

where

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

and

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

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 $x_0(n) \triangleq +1$ 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

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

vector. The optimization algorithm is given by

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

where η is the step-learning hyperparameter and $\hat{\mathbf{g}}(n) \triangleq \nabla \mathcal{E}(\mathbf{w})$ is the stochastic approximation of the gradient vector, being $\mathcal{E}(\mathbf{w})$ 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,

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

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

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

Substituting (7) into (4), it yields the learning equation, given by

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

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 accuracy² 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 testing datasets, respectively³.

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 1⁴.

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

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

⁴Since 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 */
1 forall  $\{1, 2, \dots, N_r\}$  do
2    $\mathbf{w}(n) \leftarrow \text{initialize}$ 
3    $\mathbf{X}, \mathbf{d} \leftarrow \text{shuffle}$ 
4    $(\mathbf{X}_{trn}, \mathbf{d}_{trn}), (\mathbf{X}_{tst}, \mathbf{d}_{tst}) \leftarrow \text{hold-out}$  /* training and testing dataset */
5   forall  $\{1, 2, \dots, N_e\}$  do
6     forall Instances in the training dataset do
7        $v(n) \leftarrow \mathbf{w}^\top(n) \mathbf{x}(n)$ 
8        $y(n) \leftarrow \varphi(v(n))$ 
9        $e(n) \leftarrow d(n) - y(n)$ 
10       $\mathbf{w}(n+1) \leftarrow \mathbf{w}(n) + \eta \mathbf{x}(n) e(n)$ 
11     $\mathbf{X}_{trn}, \mathbf{d}_{trn} \leftarrow \text{shuffle}$ 
12   $accuracy \leftarrow \text{test}(\mathbf{X}_{tst}, \mathbf{d}_{tst})$ 
```

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

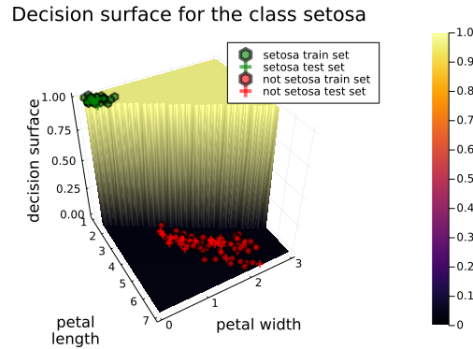


Figure 1: Decision surface of setosa class.

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.

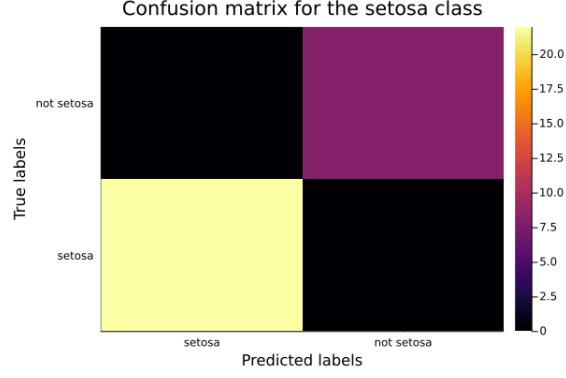


Figure 2: confusion matrix for setosa class.

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%.

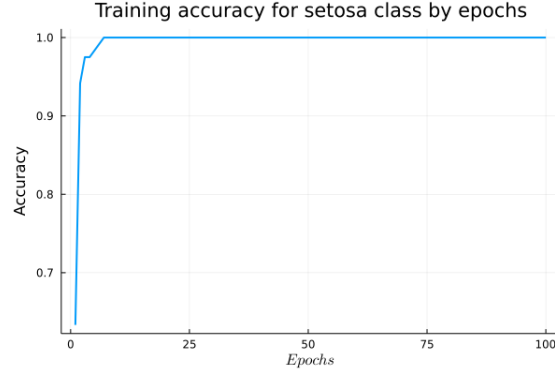


Figure 3: Training dataset evolution for the setosa classification.

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.

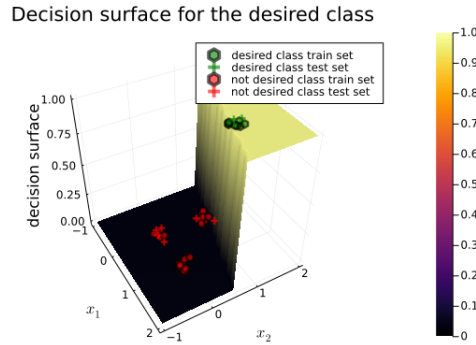


Figure 4: Decision surface for the desired class.

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

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

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

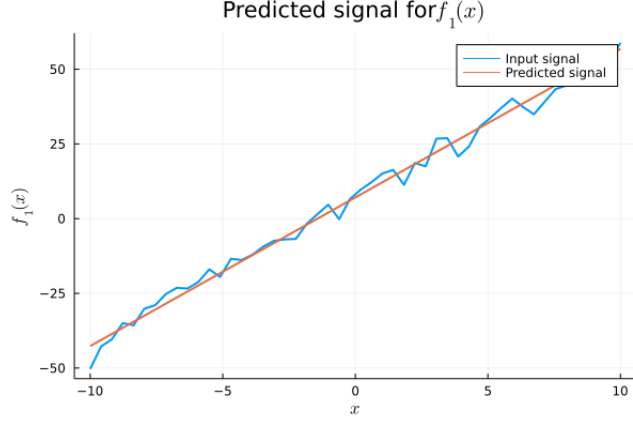
$$\begin{bmatrix} 1 & x_1(n) & x_2(n) \end{bmatrix} \in \mathbb{R}^3 \quad (9)$$

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

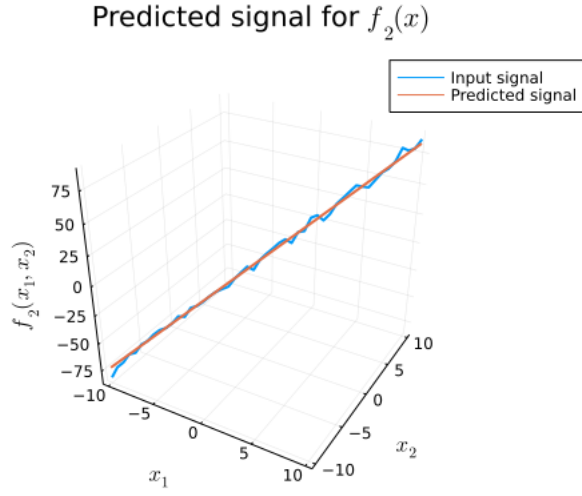
Table 2: ADALINE performance for regression problem

$f(\cdot)$	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

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

where J is the number of classes (one neuron for each class) and N_a is the number of attributes. The learning algorithm is given by

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \eta \boldsymbol{\delta}(n) \mathbf{x}(n)^\top, \quad (11)$$

where $\boldsymbol{\delta}(n) = [\delta_1(n) \ \delta_2(n) \ \dots \ \delta_J(n)]^\top$ 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

$$\mathbf{v}(n) = [v_1(n) \ v_2(n) \ \dots \ v_J(n)]^\top = \mathbf{W}(n)\mathbf{x}(n). \quad (12)$$

Notice that

$$\begin{aligned} \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} &= \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \\ &= -\delta_j(n)x_i(n), \end{aligned} \quad (13)$$

where

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = x_i(n) \quad (14)$$

is the i th input at the instant n , and

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = -\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial x_i(n)} \frac{\partial x_i(n)}{\partial v_j(n)} \quad (15)$$

is the local gradient of the j th 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 i th synaptic weight for the j th neuron at the instant n .

Note also that

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n), \quad (16)$$

$$\frac{\partial e_j(n)}{\partial x_i(n)} = -1, \quad (17)$$

and

$$\frac{\partial x_i(n)}{\partial v_j(n)} \triangleq \varphi'(v_j(n)). \quad (18)$$

By substituting the equations (16), (17), and (18) into (25), we get

$$\delta_j(n) = e_j(n)\varphi'(v_j(n)), \quad (19)$$

or in matricial notation,

$$\boldsymbol{\delta}(n) = \mathbf{e}(n) \odot \boldsymbol{\varphi}'(\mathbf{v}(n)), \quad (20)$$

where \odot is the Hadamard product and $\boldsymbol{\varphi}'(\mathbf{v}(n)) = \begin{bmatrix} \varphi'(v_1(n)) & \varphi'(v_2(n)) & \cdots & \varphi'(v_K(n)) \end{bmatrix}^\top$.

The synaptic weights update is given by

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}. \quad (21)$$

Substituting the Equation (13) into (21), we have that

$$\Delta w_{ji}(n) = \eta \delta_j(n) x_i(n), \quad (22)$$

or in matricial notation

$$\Delta \mathbf{W}(n) = \eta \boldsymbol{\delta}(n) \mathbf{x}(n)^\top. \quad (23)$$

Finally, the update function is given by

$$\begin{aligned} \mathbf{W}(n+1) &= \mathbf{W}(n) + \Delta \mathbf{W}(n) \\ &= \mathbf{W}(n) + \eta \boldsymbol{\delta}(n) \mathbf{x}(n)^\top. \end{aligned} \quad (24)$$

The Algorithm 2 summarizes the procedure of the SLP algorithm.

Algorithm 2: Single-layer perceptron

```

Input:  $\mathbf{X}, \mathbf{D}$  /* attributes and labels dataset */
1 forall  $\{1, 2, \dots, N_r\}$  do
2    $\mathbf{W}(n) \leftarrow$  initialize
3    $\mathbf{X}, \mathbf{D} \leftarrow$  shuffle
4    $(\mathbf{X}_{trn}, \mathbf{D}_{trn}), (\mathbf{X}_{tst}, \mathbf{D}_{tst}) \leftarrow$  hold-out /* training and testing dataset */
5   forall  $\{1, 2, \dots, N_e\}$  do
6     forall Instances in the training dataset do
7        $\mathbf{v}(n) \leftarrow \mathbf{W}(n) \mathbf{x}(n)$ 
8        $\mathbf{y}(n) \leftarrow \boldsymbol{\varphi}(\mathbf{v}(n))$ 
9        $\mathbf{e}(n) \leftarrow \mathbf{d}(n) - \mathbf{y}(n)$ 
10       $\boldsymbol{\delta}(n) \leftarrow \mathbf{e}(n) \odot \boldsymbol{\varphi}'(\mathbf{v}(n))$ 
11       $\mathbf{W}(n+1) \leftarrow \mathbf{W}(n) + \eta \boldsymbol{\delta}(n) \mathbf{x}(n)^\top$ 
12     $\mathbf{X}_{trn}, \mathbf{d}_{trn} \leftarrow$  shuffle
13   $accuracy \leftarrow$  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 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 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.

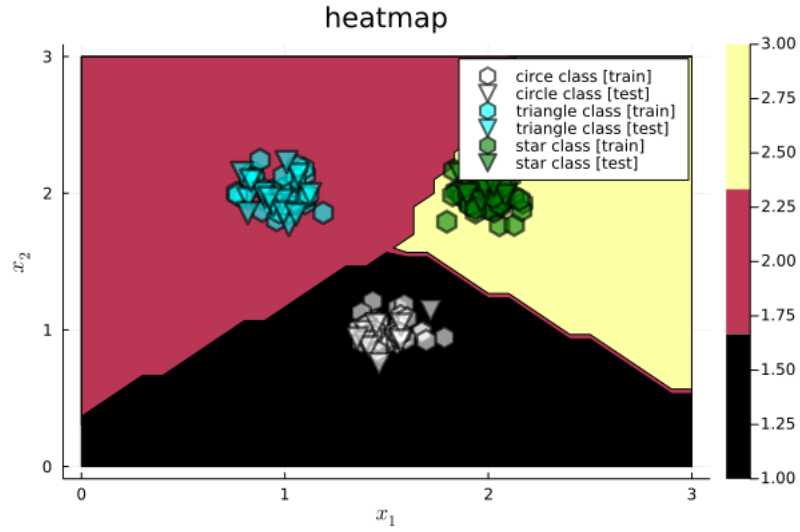


Figure 6: Heatmap of the dummy dataset.

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.

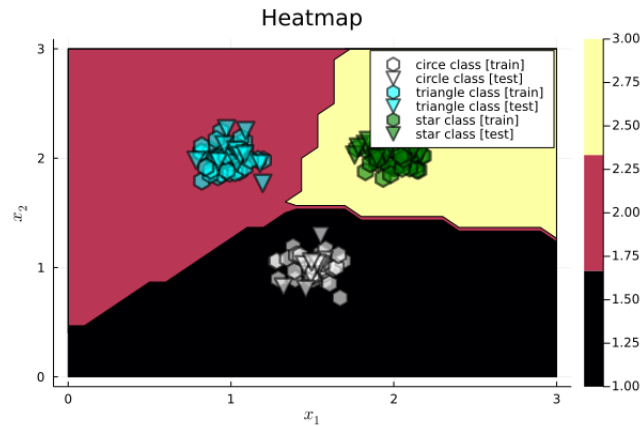
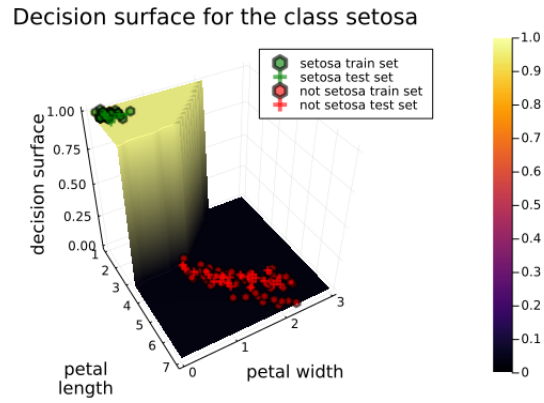


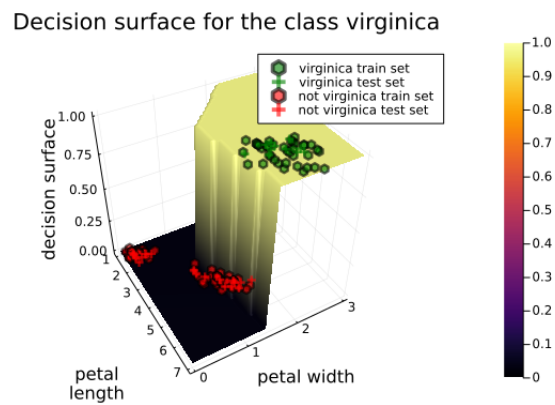
Figure 7: The dummy dataset for the SLP with logistic activation function.

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

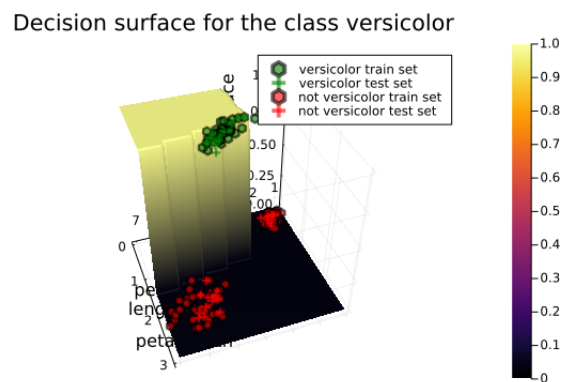
shown in Figure 8. It is possible to notice that the classifier can solve the problem for the setosa class as it is linearly separable from the other classes for the attributes considered (petal length and petal width).



(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 structure, 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 can successfully solve nonlinear problems.

The presence of hidden layers makes the learning phase more complicated to devise since one must decide how the signal error should propagate toward the first layer. A popular learning algorithm 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 derivative of the activation function of the own neuron.

Since the update equation for the output layer was already derived in the SLP algorithm⁵, we will focus on finding the update equation of the l th hidden layer. Beginning with the outmost hidden layer and recalling that the local gradient of the j th neuron on this layer is given by

$$\begin{aligned}
 \delta_j^{(L-1)}(n) &= -\frac{\partial \mathcal{E}(n)}{\partial v_j^{(L-1)}(n)} \\
 &= -\frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} \frac{\partial y_j^{(L-1)}(n)}{\partial v_j^{(L-1)}(n)} \\
 &= -\frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} \varphi'(v_j^{(L-1)}(n)), \tag{25}
 \end{aligned}$$

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

where the third equation follows that

$$\varphi'(v_j^{(L-1)}(n)) = \frac{\partial y_j^{(L-1)}(n)}{\partial v_j^{(L-1)}(n)}. \quad (26)$$

We can derive the learning equation for this layer and generalize it to all hidden layers. In this section, a superscript was added to indicate which layer the signals refer to.

Remember that $y_j^{(L-1)}(n)$ is the j th input signal on the output layer (L th layer), and that

$$\frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} = \sum_{k=1}^{m_L} e_k^{(L)}(n) \frac{\partial e_k^{(L)}(n)}{\partial y_j^{(L-1)}(n)}, \quad (27)$$

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

$$\frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} = \sum_{k=1}^{m_L} e_k^{(L)}(n) \frac{\partial e_k^{(L)}(n)}{\partial v_k^{(L)}(n)} \frac{\partial v_k^{(L)}(n)}{\partial y_j^{(L-1)}(n)}, \quad (28)$$

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

$$\frac{\partial e_k^{(L)}(n)}{\partial v_k^{(L)}(n)} = -\varphi'(v_k^{(L)}(n)). \quad (29)$$

Note that

$$v_k^{(L)} = \sum_{j=0}^{m_L} w_{kj}^{(L)}(n) y_j^{(L-1)}(n). \quad (30)$$

Therefore,

$$\frac{\partial v_k^{(L)}(n)}{\partial y_j^{(L-1)}(n)} = w_{kj}^{(L)}(n) \quad (31)$$

Substituting the Equations (29) and (31) into 28, we get

$$\begin{aligned} \frac{\partial \mathcal{E}(n)}{\partial y_j^{(L-1)}(n)} &= - \sum_{k=1}^{m_L} e_k^{(L)}(n) \varphi'(v_k^{(L)}(n)) w_{kj}^{(L)}(n) \\ &= - \sum_{k=1}^{m_L} \delta_k^{(L)}(n) w_{kj}^{(L)}(n), \end{aligned} \quad (32)$$

where the second equation follows that $\delta_k^{(L)} = e_k^{(L)}(n) \varphi'(v_k^{(L)}(n))$. Finally, substituting

Equation (32) into (25), the local gradient of the j th neuron on the $(L-1)$ th hidden layer is given by

$$\delta_j^{(L-1)}(n) = \varphi'(v_j^{(L-1)}(n)) \sum_{k=1}^{m_L} \delta_k^{(L)}(n) w_{kj}^{(L)}(n), \quad (33)$$

or in matricial notation

$$\boldsymbol{\delta}^{(L-1)}(n) = \boldsymbol{\varphi}'(\mathbf{v}^{(L-1)}(n)) \odot \mathbf{W}^{(L)}(n) \boldsymbol{\delta}^{(L)}(n), \quad (34)$$

The Equation (34) can be generalized to the l th layer:

$$\boldsymbol{\delta}^{(l)}(n) = \boldsymbol{\varphi}'(\mathbf{v}^{(l)}(n)) \odot \mathbf{W}^{(l+1)}(n) \boldsymbol{\delta}^{(l+1)}(n), \quad (35)$$

where $1 \leq l < L$. The synaptic weights update is given by

$$\Delta \mathbf{W}^{(l)}(n) = \eta \boldsymbol{\delta}^{(l)}(n) \mathbf{y}^{(l-1)}(n)^\top, \quad (36)$$

where $\mathbf{y}^{(0)}(n) \triangleq \mathbf{x}(n)$. Therefore, the learning equation of the hidden layer is given by

$$\begin{aligned} \mathbf{W}^{(l)}(n+1) &= \mathbf{W}^{(l)}(n) + \Delta \mathbf{W}^{(l)}(n) \\ &= \mathbf{W}^{(l)}(n) + \eta \boldsymbol{\delta}^{(l)}(n) \mathbf{y}^{(l-1)}(n)^\top. \end{aligned} \quad (37)$$

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    $\mathbf{W}(n) \leftarrow \text{initialize}$ 
3    $\mathbf{X}, \mathbf{D} \leftarrow \text{shuffle}$ 
4    $(\mathbf{X}_{trn}, \mathbf{D}_{trn}), (\mathbf{X}_{tst}, \mathbf{D}_{tst}) \leftarrow \text{hold-out}$  /* training and testing dataset */
5   forall  $\{1, 2, \dots, N_e\}$  do
6     forall Instances in the training dataset do
7       /* forward phase */
8       for  $l \in \{1, 2, \dots, L\}$  do
9          $\mathbf{v}^{(l)}(n) \leftarrow \mathbf{W}^{(l)}(n) \mathbf{y}^{(l-1)}(n)$ 
10         $\mathbf{y}^{(l)}(n) \leftarrow \boldsymbol{\varphi}(\mathbf{v}^{(l)}(n))$ 
11        /* backward phase */
12         $\mathbf{e}(n) \leftarrow \mathbf{d}(n) - \mathbf{y}^{(L)}(n)$  /* output layer */
13         $\boldsymbol{\delta}^{(L)}(n) \leftarrow \mathbf{e}(n) \boldsymbol{\varphi}'(\mathbf{v}^{(L)}(n))$ 
14         $\mathbf{W}^{(L)}(n+1) \leftarrow \mathbf{W}^{(L)}(n) + \eta \boldsymbol{\delta}^{(L)}(n) \mathbf{y}^{(L-1)}(n)^\top$ 
15        for  $l \in \{L-1, L-2, \dots, 1\}$  do
16           $\boldsymbol{\delta}^{(l)}(n) = \boldsymbol{\varphi}'(\mathbf{v}^{(l)}(n)) \odot \mathbf{W}^{(l+1)}(n) \boldsymbol{\delta}^{(l+1)}(n)$ 
17     $\mathbf{X}_{trn}, \mathbf{d}_{trn} \leftarrow \text{shuffle}$ 
18   $\text{accuracy} \leftarrow \text{test}(\mathbf{X}_{tst}, \mathbf{d}_{tst})$ 
```
