Chapter 4

Multidimensional Radial Wavelon - Feed-Forward Wavelet Neural Network (MRW-FFWNN)

In this chapter, we present the description of the proposed classifier called MRW-FFWNN and its training algorithm. Architectures and characteristics of other classifiers based on neural networks, which were used in this thesis for comparison purposes, are also described.

4.1 Description of the MRW-FFWNN.

The proposed classifier is composed of processing nodes known as Multidimensional Radial Wavelon (MRW), which were first proposed in [ZHA92], [ZHA93] in a forward propagation architecture, designed for the approximation of multivariable functions. Figure 4.1 shows the architecture of a MRW unit, which is composed of two modules. The first module contains a radial function and is denoted by the letter R; the second

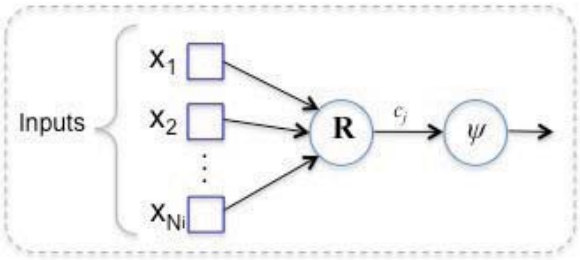

Figure 4.1: The MRW unit.

contains a one-dimensional wavelet function denoted by ψ. A wavelet function ψ is radial if it has the form $\psi(x) = \phi(||x||)$. The purpose of this division is to process the input vector so that they can be used for a one-dimensional wavelet function.

Figure 4.2 shows the architecture of the proposed MRW-FFWNN model. It is composed of three layers. *Layer 1* is an input layer, which transmits input values to the next layer directly. The external inputs are represented by x_k, where $k = 1...N_i$, N_i is the number of inputs to the network. *Layer 2* has MRW units. Each node of this layer has a radial function and an one-dimensional wavelet function denoted by ψ. Outputs of the R units are denoted by c_j, $j = 1...N_w$, where N_w is the number of multidimensional radial wavelon units. The outputs of the MRW units are $\psi(c_j)$ where c_j is given by:

$$c_j = ||\mathbf{d_j}(\mathbf{x} - \mathbf{t}_j)|| \tag{4.1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{t}_j$ and $\mathbf{d_j}$ are the translation and dilation factor of the wavelets, respectively. The difference between our proposal and the architectures proposed in [ZHA92], [ZHA93] is that ours use the same dilation factor for all dimensions of each neuron. Therefore, there are N_w parameters of dilation into all the network, where N_w is the number of neurons, whereas in [ZHA92], [ZHA93] there are $N_i \times N_w$ dilation factors, where N_i is the number of inputs (dimensions). The advantage of our proposal is to reduce the number of adjusting parameters in the learning process of the net. *Layer 3* is an output layer composed of a linear combiner.

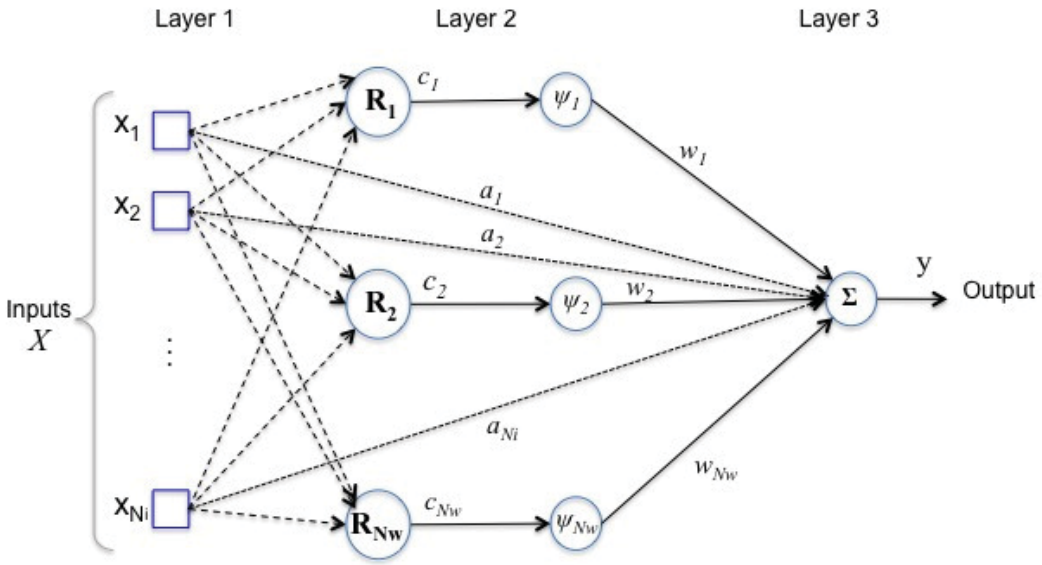

Figure 4.2: The proposed MRW-FFWNN structure.

Direct connections from the input layer to the output layer are weighted by a_k, $k = 1, ..., N_i$, where N_i is the number of inputs. Synaptic weights between the output and the wavelet units are denoted by w_j. The input layer receives the external inputs and transmits them directly to the multidimensional radial layer. It is posible to denote the vector $\mathbf{x_k}$ of external inputs by means of

$$\mathbf{x_k} = [x_1, x_2, ..., x_{N_i}] \tag{4.2}$$

where N_i is the number of external inputs.

Each MRW manages its own translation t_j and dilation parameters d_j for each of inputs in $\mathbf{x_k}$. Parameters $\mathbf{d_j}$ and $\mathbf{t_j}$ can be represented as:

$$\mathbf{d_j} = [d_{j,1}, d_{j,2}, ..., d_{j,N_i}] \tag{4.3}$$

$$\mathbf{t_j} = [t_{j,1}, t_{j,2}, ..., t_{j,N_i}] \tag{4.4}$$

$$j = 1, 2, ...N_w \tag{4.5}$$

where N_w represents the number of multidimensional radial wavelon units found in this layer. The principal task of the units R_j, $j = 1, ..., N_w$, within this layer is to process the multiple inputs in order to produce a scalar value, which serves in the construction of the inputs to the wavelet unit [ALA14].

To calculate the outputs of units R_j, it is useful to define:

$$A(\mathbf{x_k}, \mathbf{d_j}, \mathbf{t_j}) = diag(\mathbf{d_j})(\mathbf{x_k} - \mathbf{t_j})^T \tag{4.6}$$

where T denotes the transposed operation of the difference between the inputs vector $\mathbf{x_k}$ and the translation vector $\mathbf{t_j}$; $diag(\mathbf{d_j})$ represents a diagonal matrix constructed from the dilation vector $\mathbf{d_j}$, [ALA14]:

$$diag(\mathbf{d_j}) = \begin{pmatrix} d_{1,1} & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & d_{j,N_i} \end{pmatrix} \tag{4.7}$$

The output of each unit R_j is given by:

$$R_j(\mathbf{x_k}, \mathbf{d_j}, \mathbf{t_j}) = \left[A(\mathbf{x_k}, \mathbf{d_j}, \mathbf{t_j})^T A(\mathbf{x_k}, \mathbf{d_j}, \mathbf{t_j})\right]^{1/2} \tag{4.8}$$

$$c_j = R(\mathbf{x_k}, \mathbf{d_j}, \mathbf{t_j}) \tag{4.9}$$

The output layer is made of a linear combiner [ALA14]. The output has a weight associated with each unit of the previous layer (w_j) and with each input unit (a_k), that is:

$$a_k, \quad k = 1, 2, ..., N_i \tag{4.10}$$

$$w_j, \quad j = 1, 2, ..., N_w \tag{4.11}$$

where N_i represents the number of inputs and N_w the number of wavelet units in the architecture. Finally the output of the wavelet unit is given by $\psi(c_j)$. Therefore, the MRW-FFWNN output y is given by:

$$y = \sum_{j=1}^{N_w} w_j \psi_j(c_j) + \sum_{k=1}^{N_i} a_k x_k \tag{4.12}$$

where, x_k is the input, w_j is the connection weight between MRW nodes and output nodes, and a_k is the connection weight between the input nodes and the output node.

4.2 Learning algorithm of the MRW-FFWNN.

The architecture of the proposed model MRW-FFWNN has been defined, now we describe the learning algorithm used. This is based on a gradient-descent strategy, whose goal is to minimize the quadratic cost function[SUN07]:

$$E(n) = \frac{1}{2}[y_d(n) - y(n)]^2 = \frac{1}{2}e^2(n) \tag{4.13}$$

where, $y_d(n)$ is the desired output and $y(n)$ is the actual output of MRW-FFWNN at time n.

The tuning parameters $W = [a_k \ t_{jk} \ d_j \ w_j]$ of the MRW-FFWNN are continually adjusted to minimize the error, which is obtained after a certain number of training cycles. The delta rule determines the amount of update for the tuning parameters, based on the gradient direction along with a learning rate η, as follows:

$$W(n+1) = W(n) + \Delta W(n) = W(n) + \eta \left(-\frac{\partial E}{\partial W(n)} \right) \tag{4.14}$$

The partial derivative of the cost function with respect to the tuning parameters is given by

$$\frac{\partial E}{\partial W(n)} = \sum\nolimits_{k\in 0} e_k \frac{\partial y(n)}{\partial W(n)} \tag{4.15}$$

It is required to calculate $\frac{\partial y(n)}{\partial W(n)}$ for each one of the tuning parameters that belongs to the vector $W = [a_k\ t_{jk}\ d_j\ w_j]$. Now, such derivatives are shown.

1. **Weights between the input layer and output layer**

   The partial derivative of $\frac{\partial y(n)}{\partial a_k}$ is computed by using the output of the system as follows:

$$\frac{\partial y(n)}{\partial a_k} = \frac{\partial \sum_{j=1}^{N_w} w_j \psi_j(c_j) + \sum_{k=1}^{N_i} a_k x_k}{\partial a_k} \tag{4.16}$$

   The weights between the output of MRW nodes and the output of the MRW-FFWNN do not depend on a_k. Therefore the previous equation is converted into

$$\frac{\partial y(n)}{\partial a_k} = \frac{\partial \sum_{k=1}^{N_i} a_k x_k}{\partial a_k} = x_k \tag{4.17}$$

   Equation (4.17) is the partial derivative of the output with respect to a_k in the MRW-FFWNN model.

2. **The translation parameters**

   Let:

$$\frac{\partial y(n)}{\partial t_{jk}} = \frac{\partial \sum_{j=1}^{N_w} w_j \psi_j(c_j) + \sum_{k=1}^{N_i} a_k x_k}{\partial t_{jk}}$$

In this equation, the second term in the addition does not depend on the parameters of translation of the units of the wavelet layer, so that it is converted into

$$\frac{\partial y(n)}{\partial t_{jk}} = \frac{\partial \sum_{j=1}^{N_w} w_j \psi_j(c_j)}{\partial t_{jk}} = w_j \frac{\partial \psi(c_j)}{\partial t_{jk}}$$

Applying the chain rule, we obtain:

$$\frac{\partial y(n)}{\partial t_{jk}} = w_j \frac{\partial \psi(c_j)}{\partial c_j} \frac{\partial c_j}{\partial t_{jk}}$$

Developing the third term of the product we have:

$$\frac{\partial y(n)}{\partial t_{jk}} = w_j \frac{\partial \psi(c_j)}{\partial c_j} \frac{\partial \left[d_j^2 (x_1 - t_{j1})^2 + ... + d_j^2 (x_k - t_{jk})^2 \right]^{\frac{1}{2}}}{\partial t_{jk}}$$

which gives

$$\frac{\partial y(n)}{\partial t_{jk}} = w_j \frac{\partial \psi(c_j)}{\partial c_j} \left[\frac{1}{2c_j} (-2d_j^2 x_k + 2d_j^2 t_{jk}) \right] \tag{4.18}$$

with $c_j = R(\mathbf{x_k}, \mathbf{d_j}, \mathbf{t_j})$.

Equation (4.18) is the partial derivative of the output with respect to t_{jk} in the MRW-FFWNN model.

3. **The dilation parameters**

Likewise it can be established:

$$\frac{\partial y(n)}{\partial d_j} = \frac{\partial \sum_{j=1}^{N_w} w_j \psi_j(c_j) + \sum_{k=1}^{N_i} a_k x_k}{\partial d_j}$$

In this equation, the second term in the addition does not depend on the parameters of dilation of the units of the wavelet layer, so it is converted into

$$\frac{\partial y(n)}{\partial d_j} = \frac{\partial \sum_{j=1}^{N_w} w_j \psi_j(c_j)}{\partial d_j} = w_j \frac{\partial \psi(c_j)}{\partial d_j}$$

By the chain rule, we obtain

$$\frac{\partial y(n)}{\partial d_j} = w_j \frac{\partial \psi(c_j)}{\partial c_j} \frac{\partial c_j}{\partial d_j}$$

Developing the third term of the product we have,

$$\frac{\partial y(n)}{\partial d_j} = w_j \frac{\partial \psi(c_j)}{\partial c_j} \frac{\partial \left[d_j^2 (x_1 - t_{j1})^2 + ... + d_j^2 (x_k - t_{jk})^2 \right]^{\frac{1}{2}}}{\partial d_j}$$

Therefore using Equation (4.1),

$$\frac{\partial y(n)}{\partial d_j} = w_j \frac{\partial \psi(c_j)}{\partial c_j} \left[\frac{d_j}{c_j} \left((x_1 - t_{j1})^2 + ... + (x_k - t_{jk})^2 \right) \right] \tag{4.19}$$

with $c_j = R(\mathbf{x_k}, \mathbf{d_j}, \mathbf{t_j})$.

Equation (4.19) is the partial derivative of the output with respect to d_j in the MRW-FFWNN model.

4. **Weights of the connections between the MRW nodes and the output layer**

Let:

$$\frac{\partial y(n)}{\partial w_j} = \frac{\partial \sum_{j=1}^{N_w} w_j \psi_j(c_j) + \sum_{k=1}^{N_i} a_k x_k}{\partial w_j}$$

The partial derivative is calculated as follows:

$$\frac{\partial y(n)}{\partial w_j} = \frac{\partial \sum_{j=1}^{N_w} w_j \psi_j(c_j)}{\partial w_j}$$

$$\frac{\partial y(n)}{\partial w_j} = \psi_j(c_j) \tag{4.20}$$

Table 4.1: Partial derivatives for the proposed model MRW-FFWNN.

Partial derivative of the output $\partial y(n)$ with respect to	Proposed model MRW-FFWNN
$\partial a_k(n)$	$\frac{\partial y(n)}{\partial a_k} = x_k$
$\partial t_{jk}(n)$	$\frac{\partial y(n)}{\partial t_{jk}} = w_j \frac{\partial \psi(c_j)}{\partial c_j} \left[\frac{1}{2c_j}(-2d_j^2 x_k + 2d_j^2 t_{jk}) \right]$
$\partial d_j(n)$	$\frac{\partial y(n)}{\partial d_j} = w_j \frac{\partial \psi(c_j)}{\partial c_j} \left[\frac{d_j}{c_j} \left((x_1 - t_{j1})^2 + ... + (x_k - t_{jk})^2 \right) \right]$
$\partial w_j(n)$	$\frac{\partial y(n)}{\partial w_j} = \psi_j(c_j)$

Equation (4.20) is the partial derivative of the output with respect to w_j in the MRW-FWWNN model. The adjustments to each of the tuning parameters that belong to the vector W are summarized in Table 4.1.

4.2.1 Pseudocode of learning algorithm.

Next is the pseudocode of the proposed learning algorithm (see Algorithm 1) based on a gradient-descent strategy, which establishes its general operation. The algorithm starts by defining the values of inputs number N_i and wavelons number N_w and initializing the weights $w_j, a_k, t_{jk}, and d_j$. Later on, evaluate the sample, calculate the error and update parameters (weights). Finally, it calculates the Mean squared error (MSE). The stopping conditions of the algorithm are defined by the epochs number N_{Epochs}, and the MSE_{max} defined by the user. Therefore, if the calculated value of MSE is less than maximum allowed of MSE defined as MSE_{max} or if the training cycles number exceeds the defined value as N_{Epochs}, the learning process is stopped. A flowchart of the main actions of the learning algorithm is shown in Figure 4.3.

Algorithm 1 Learning algorithm

Set value of N_i and N_w

Initialize w_j, a_k according to the best results

Initialize t_{jk}, d_j

while MSE $> MSE_{max}$ and Epochs number $< N_{Epochs}$ **do**

for $x = 1, ...,$ Samples number **do**

Evaluate (sample x)

Calculate Error (sample x)

Update parameters a_k, t_{jk}, d_j and w_j by

$Parameter + Learning\_rate \times Error \times \partial\, Output / \partial\, Parameter$

end for

Calculate MSE

end while

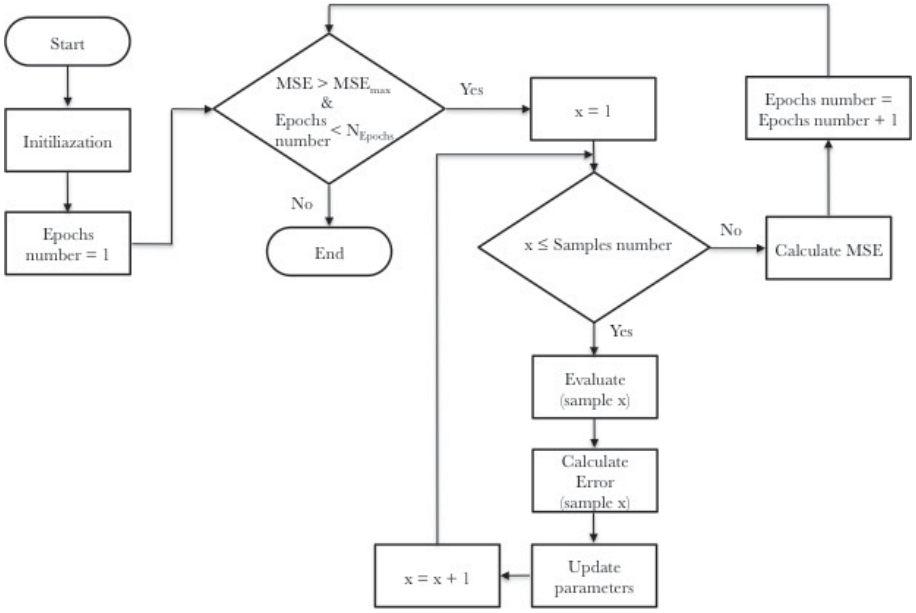

Figure 4.3: Flowchart of the learning algorithm.

4.3 Mother Wavelet Functions.

We investigated the effectiveness of three mother wavelet functions: Mexican hat wavelet, derivative of the Gaussian wavelet and Morlet wavelet as activation functions on WNN, when used to identify the EEG signals related to epilepsy. These activation functions were chosen because they are smooth and bounded continuous functions. The term "bounded" means that the output never reaches a predefined values, regardless of the input. So, the activation output remains bounded even if the net input to a neuron is large. Continuity of the functions implies that there are no sharp peaks or gaps in the function, so that they can be differentiated throughout, making it possible to implement the delta rule to adjust both input-hidden and hidden-output layer weights in the training algorithm [SAM07].

The experiments based on WNN were evaluated with these three mother wavelets functions; equations (4.21), (4.22) and (4.23) correspond to Mexican hat wavelet, Derivative of Gaussian wavelet and Morlet wavelet, respectively, and their waveforms are shown in Figure 4.4.

$$mexh(x) = \frac{2(1-x^2)e-\frac{x^2}{2}}{\pi^{\frac{1}{4}}\sqrt{3}} \tag{4.21}$$

$$\phi(x) = -xe^{(-\frac{1}{2}x^2)} \tag{4.22}$$

$$morl(x) = e^{-\frac{x^2}{2}}cos(5x) \tag{4.23}$$

The wavelet functions selected for the experiments reported here present the following characteristics: symmetry (useful in avoiding de-phasing), compact support (allow efficient implementation), orthogonality (allow fast algorithm), and bi-orthogonal (provides phase linearity). The Mexican hat and Derivative of Gaussian

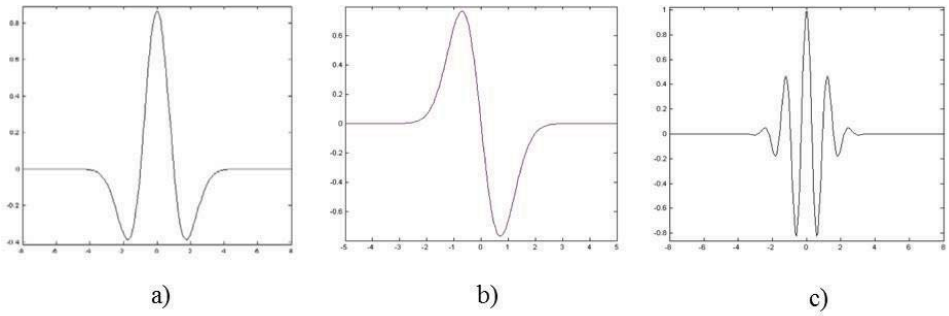

Figure 4.4: Mother Wavelet functions. a) Mexican hat wavelet. b) Derivative of the Gaussian wavelet. c) Morlet wavelet.

wavelet have a effective support of [-5, 5] while Morlet has one of [-4, 4]. Further details of these mother wavelets can be found in [MAT14], [GAN11].

4.4 Classifiers based on neural networks and wavelet-based neural networks.

Six architectures of classifiers based on neural networks and wavelet-based neural networks such as FF-ANN, Elman Network, FFWNN, MRW-FFWNN (proposed model), SRWNN and MRW-SRWNN were used in this research to classify three classes of EEG signals.

One of the major difficulties when using neural networks is the selection of the proper size, topology and parameters of the networks. There are some techniques for finding the optimal set of features of a network, such as the Differential Evolution Technique [LAH09], evolutionary algorithms [AVI14], genetic algorithms [SAN15] or existing sources [HUN12]. However, considering the rather small size of the networks tested in these experiments, we did a direct search of the best parameters of each classifier, to obtain the most suitable architecture to classify the EEG signals. We tested different combinations of parameters of each classifier, which are detailed next. Parameters include the number of hidden nodes, learning rates, activation

functions and number of training epochs to obtain the best combination possible in each architecture that provide the best result, as in [SUB05], [TZA09], [GHO07]. The experiments reported here were implemented using Matlab 2010a.

The number of hidden nodes determines the learning ability of the neural network. A network with too few hidden nodes would be incapable of differentiating between complex patterns leading to only a linear estimate of the actual trend. In contrast, if the network has too many hidden nodes it will follow the noise in the data due to over-parameterization, leading to poor generalization for untrained data [HAG10]. Due the problem of over-parameterization, it should use the minimum number of hidden neurons with which the network can be represented properly. This is achieved by evaluating of the performance of different architectures based on the results obtained with the validation group [SAM07], [HAG10].

The learning rate is a constant between 0 and 1, that adjusts how fast learning should take place. Smaller values indicate a slower weight adjustment, requiring a longer period of time to complete training; larger values accelerate the rate of weight increments [HAG10]. Accelerated weight adjustment is not necessarily better because it may cause the solution to oscillate around the optimum, leading to instability. Therefore, it is recommended to choose a learning rate as large as possible without causing large oscillations. In general, the value of the learning rate is usually between 0.05 and 0.5 [SAM07], [HAG10].

In order to find the best number of hidden nodes for the FF-ANN and Elman networks, tests were done using 6, 9, 12, 15, 16, 18, 21 and 24 nodes in the hidden layer of the network. Levenberg-Marquardt algorithm was the training method used for the FF-ANN, whereas gradient descent was the training method used for the Elman network. The stopping conditions of the learning algorithm are defined by the N_{Epochs} and the MSE_{max}; these values were chosen by observing the behavior of MSE in each network. After tests done using values of 0.05, 0.1, 0.2, 0.3, 0.4 and 0.5 as learning rates, the best effectiveness of the networks were obtained with a learning

rate of 0.05 and the maximum allowed value of Mean Square Error (MSE_{max}) was set to 0.01, whereas the maximum number of training epochs was fixed at 1000. Therefore, if the calculated value of MSE is less than 0.01 or if the epochs number exceeds the value of 1000, the training process is stopped. These values were chosen by observing the behavior of MSE in each network. The experiments done with the FFANN and ELMAN networks used sigmoid and hyperbolic tangent as activation function.

The classifiers based on WNN (FFWNN, MRW-FFWNN, SRWNN and MRW-SRWNN) were used in binary-tree strategies and OVO decomposition strategies with two aggregation methods (VOTE and WV) to classify three classes of EEG signals. Gradient descent was the training method used to the classifiers FFWNN, MRW-FFWNN, SRWNN and MRW-SRWNN. In order to find the best parameters, tests were done using values of 0.001, 0.01 and 0.1 as learning rates; 20, 40, 60, 80, 100, 150 and 200 as number of neurons; and Mexican hat, Gaussian, and Morlet wavelets as activation functions. After an exhaustive search of the architecture parameters, the best effectiveness of the networks were obtained with 60 neurons, a learning rate of 0.001and Mexican hat wavelet as activation function, in each classifier. The MSE_{max} was set to 0.1 whereas N_{Epochs} was fixed at 100. Therefore, if the calculated value of MSE is less than 0.1 or if the epochs number exceeds the value of 100, the training process is stopped. The selection of the most suitable parameters of each classifier was based on monitoring the variation of error, as in [SUB05], [TZA09], [GHO07].

Table 4.2 presents a summary of the parameters used to implement them and the computational complexity of each classifier. Computational complexity measures the amount of computational resources, such as time and space, that are needed to compute a function. The computational cost for basic operations such as arithmetic, assignment, logic tests, reading and writing are considered a constant cost. Any algorithm that does not have cycles, it has a constant computational cost, since the number of instructions required to execute that is simply a constant [COR09].

Table 4.2: Parameters for classifiers used to identify EEG signals.

Classifier	Input/Output nodes	Hidden nodes	Activation function	MSE_{max}	Learning rate	Training epochs
(1) FF-ANN	6 / 3	6 ... 24	Sigm., Hyp. Tan.	0.01	0.05	1000
(2) ELMAN	6 / 3	6 ... 24	Sigm., Hyp. Tan.	0.01	0.05	1000
(3) FFWNN	6 / 1	60	Mexican hat	0.1	0.001	100
(4) MRW-FFWNN	6 / 1	60	Mexican hat	0.1	0.001	100
(5) SRWNN	6 / 1	60	Mexican hat	0.1	0.001	100
(6) MRW-SRWNN	6 / 1	60	Mexican hat	0.1	0.001	100

Whereas that an algorithm with a cycle, which goes from 1 to n, it has a computational cost of $\mathcal{O}(n)$, since the algorithm executes a fixed number of instructions before the cycle, a fixed number of instructions after the cycle and a constant number of instructions inside the cycle that executes n times. An algorithm with a cycle inside another cycle has a computational cost of $\mathcal{O}(n^2)$ [COR09]. Therefore, according the pseudocode in Section 4.2.1, the cost of the proposed model MRW-FWNN is $\mathcal{O}(n^2)$. The computational complexity of each classifier used in this work is also $\mathcal{O}(n^2)$ [ALA14], [GAR11]. The characteristics of each architecture used as classifier are shown below.

4.4.1 Classifier 1: FF-ANN.

This is the simplest neural network able to generate non-linear decision regions. A FF-ANN with 6 input nodes (one for each feature), one hidden layer and 3 output nodes (one for each class) was used as classifier, using the code provided by [GOM09]. In order to find the best number of hidden nodes for this classifier, we tested with 6 to 24 hidden nodes. Figure 4.5 shows this network architecture.

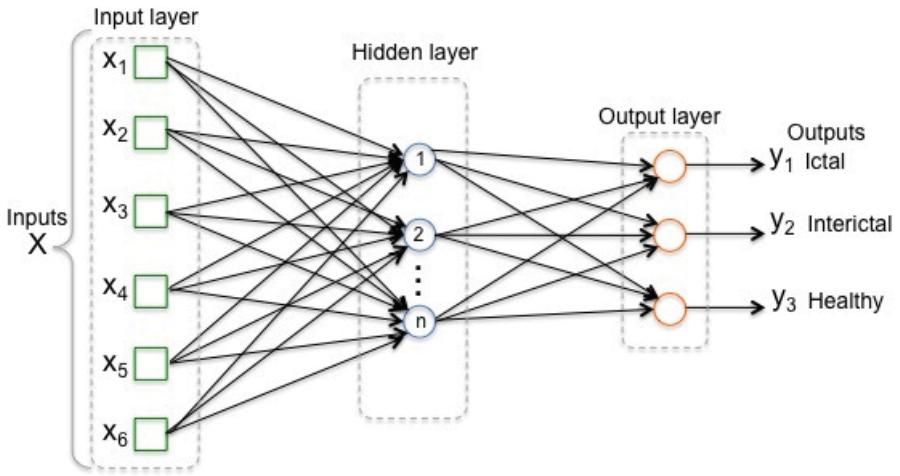

Figure 4.5: FF-ANN architecture.

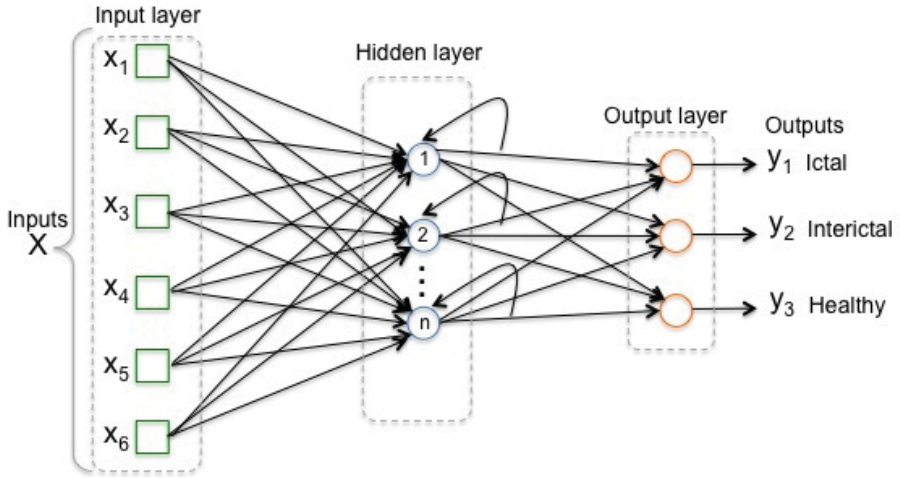

Figure 4.6: Elman Network architecture.

4.4.2 Classifier 2: Elman Network.

In an Elman network, hidden neuron outputs are fed back as inputs to the hidden neurons at the next time step. The recurrent connections facilitate encapsulation of the longterm memory structure of the data [SAM07]. Here, we modified the code provided by [GOM09], to build an Elman Network [SAM07] for the classification of three classes of EEG signals. Figure 4.6 shows the Elman Network architecture, which has 6 input nodes (one for each feature) and 3 output nodes (one for each class); we evaluated this classifier varying from 6 to 24 hidden nodes.

4.4.3 Classifier 3: FFWNN.

Wavelet neural networks combine the theory of wavelets and neural networks into one. A wavelet neural network generally consists of a feed-forward neural network, with one hidden layer, whose activation functions are drawn from an orthonormal wavelet family [ZHA92]. The FFWNN architecture used for our experiments is shown in Figure 4.7; it has 6 input nodes (one for each feature) and 1 output node. Sixty nodes were used in layer 2 and 3. As we stated before, Mexican hat function, Gaussian function and Morlet function were used as activation functions in the layer 2 of the FFWNN, although the Mexican hat gave the best effectiveness. The output of the FFWNN is given by [ZHA92]

$$y = \sum_{j=1}^{N_w} w_j \Psi_j(x) + \sum_{k=1}^{N_i} a_k x_k \tag{4.24}$$

and,

$$\Psi_j(x) = \prod_{k=1}^{N_i} \psi(u_{jk}) \tag{4.25}$$

where, x_k is the input, w_j is the connection weight between product nodes and output nodes, a_k is the connection weight between the input nodes and the output node, u_{jk} are the outputs of the wavelet mother, the subscript jk indicates the k-th input term of the j-th wavelet, and Ψ_j is the product of the mother wavelets.

4.4.4 Classifier 4: MRW-FFWNN.

The MRW-FFWNN structure is our proposed model, which is explained in Section 4.1 (see Figure 4.2). The network used in the experiments shown here has 6 input nodes (one for each feature) and 1 output node. Mexican hat function, Gaussian function and Morlet function were used as activation functions in the layer 2 of this structure, finding out that the Mexican hat gave the best effectiveness.

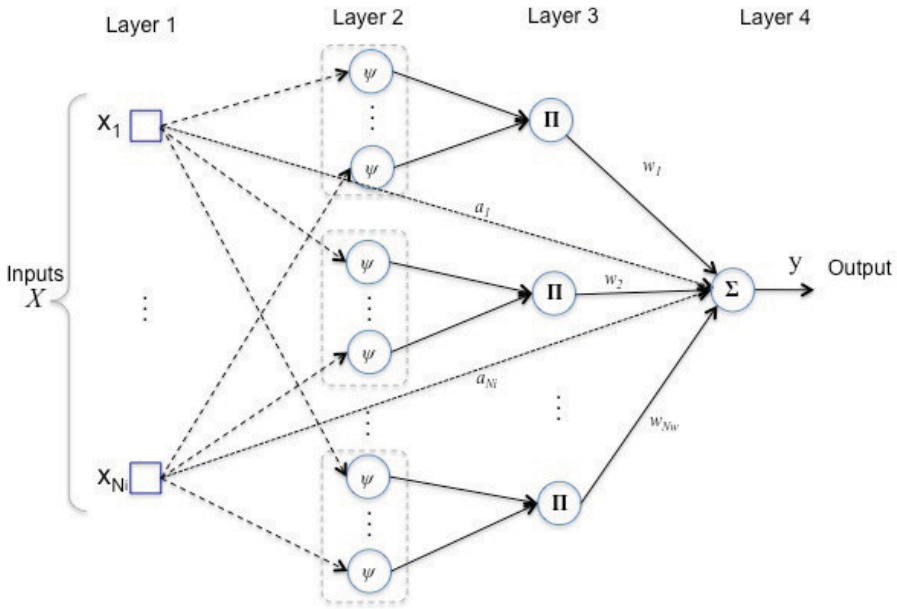

Figure 4.7: The FFWNN architecture.

4.4.5 CLASSIFIER 5: SRWNN.

A variant of the FFWNN architecture is the SRWNN [SUN05], [SUN07]. The SRWNN architecture has the same structure that FFWNN with an additional feedback loop in layer 2. The SRWNN is shown in Figure 4.8 and it has 6 input nodes (one for each feature) and 1 output node. Mexican hat function, Gaussian function and Morlet function were used as activation functions in layer 2 of the SRWNN, although the Mexican hat gave the best effectiveness. The output of each node is given by [SUN05], [SUN07]

$$y = \sum_{j=1}^{N_w} w_j \Psi_j(x) + \sum_{k=1}^{N_i} a_k x_k \tag{4.26}$$

and,

$$\Psi_j(x) = \prod_{k=1}^{N_i} \psi(\mu_{jk}) \tag{4.27}$$

$$\mu_{jk}(n) = x_k(n) + \psi_{jk}(n-1)\alpha_{jk} \tag{4.28}$$

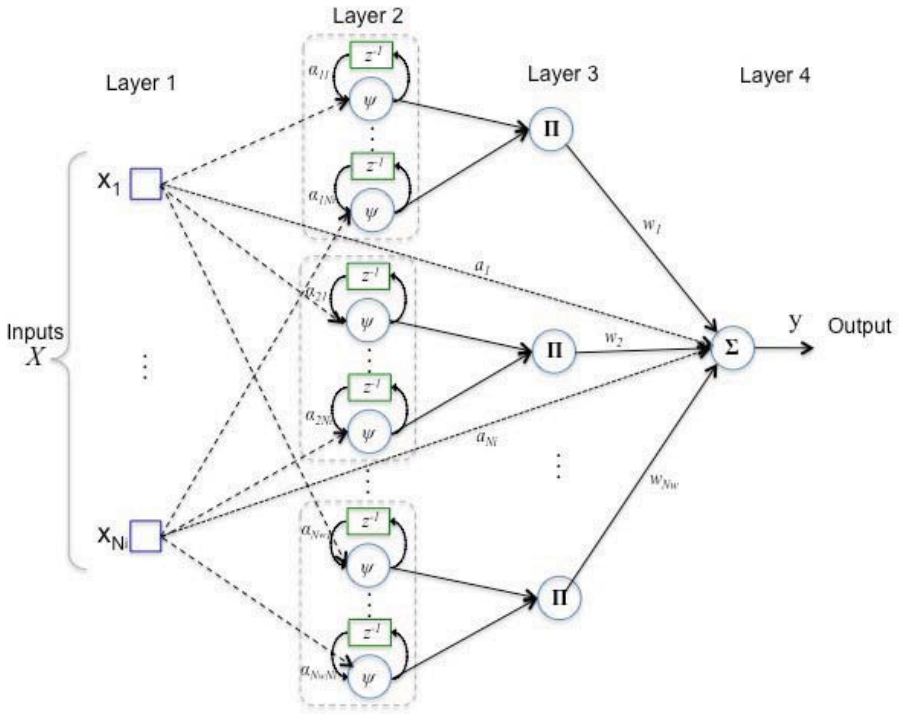

Figure 4.8: The SRWNN architecture [SUN05].

where, x_k is *j-input* to the node, α_{jk} denotes the weight of the self-feedback loop, w_j is the connection weight between product nodes and output nodes, a_k is the connection weight between the input nodes and the output node, and μ_{jk} are the outputs of the wavelet mother, the subscript jk indicates the k-th input term of the j-th wavelet.

4.4.6 Classifier 6: MRW-SRWNN.

The MRW-SRWNN has 6 input nodes (one for each feature) and 1 output node (see Figure 4.9), further details of this structure can be found in [ALA14], [SUN07]. Mexican hat function, Gaussian function and Morlet function were used as activation functions in layer 2 of the MRW-SRWNN, although the Mexican hat gave the best effectiveness.

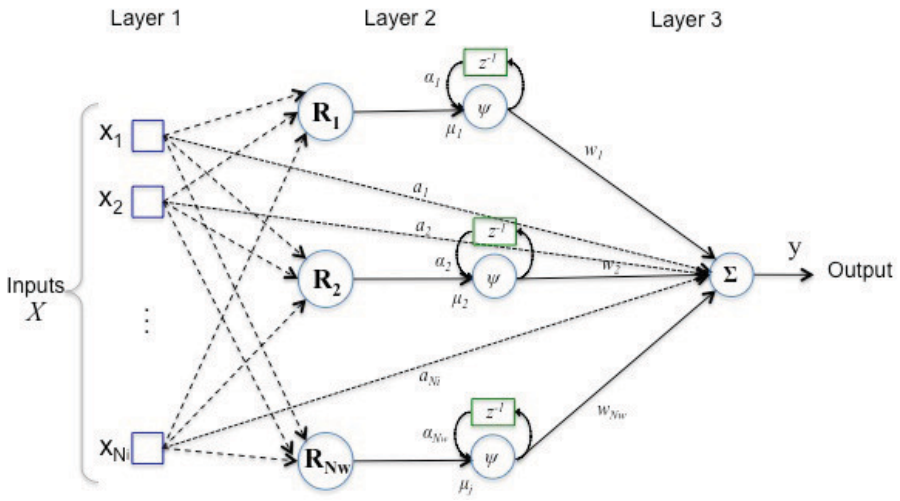

Figure 4.9: The MRW-SRWNN architecture [ALA14], [SUN07].

4.5 Discussion.

In this chapter, we presented the description of the proposed classifier called MRW-FFWNN and its learning algorithm. The MRW-FFWNN is composed of three layers. Layer 1 is an input layer, which transmits input values to the next layer directly. Layer 2 has MRW units. Each node of this layer has a radial function and an one-dimensional wavelet function. The purpose of this layer is to process the input vector to be used by a one-dimensional wavelet function. Layer 3 is an output layer composed of a linear combiner. The difference between our proposal and the architectures proposed in [ZHA92], [ZHA93] is that ours use the same dilation factor for all dimensions of each neuron. This modification reduces the number of adjusting parameters in the net. The computational complexity of [ZHA92] and [ZHA93] is $\mathcal{O}(n^2)$ similar to our proposed model and a classical sigmoid neural network. The training method for MRW-FFWNN is based on a gradient-descent strategy. The tuning parameters $W = [a_k \ t_{jk} \ d_j \ w_j]$ of the MRW-FFWNN are continually adjusted to minimize the error, which is obtained after a certain number of training cycles. In this chapter we also explain that the computational complexity of the proposed model MRW-FWNN is $\mathcal{O}(n^2)$, considering that the computational complexity measures the amount of computational resources, such as time and space, that are

needed to compute a function. The characteristics of the mother wavelet functions used as activation function also were presented in this chapter. Architectures and characteristics of other classifiers based on neural networks and Wavelet-Based Neural Networks which were used in this thesis for comparison purposes were also described.