3

Orthonormal Basis Lattice Neural Networks

Angelos Barmpoutis[1] and Gerhard X. Ritter[2]

[1] University of Florida, Gainesville, FL 32611, USA
abarmpou@cise.ufl.edu
[2] University of Florida, Gainesville, FL 32611, USA
ritter@cise.ufl.edu

Summary. Lattice based neural networks are capable of resolving some difficult non-linear problems and have been successfully employed to solve real-world problems. In this chapter a novel model of a lattice neural network (LNN) is presented. This new model generalizes the standard basis lattice neural network (SB-LNN) based on dendritic computing. In particular, we show how each neural dendrite can work on a different orthonormal basis than the other dendrites. We present experimental results that demonstrate superior learning performance of the new Orthonormal Basis Lattice Neural Network (OB-LNN) over SB-LNNs.

3.1 Introduction

The artificial neural model which employs lattice based dendritic computation has been motivated by the fact that several researchers have proposed that dendrites, and not neurons, are the elementary computing devices of the brain, capable of implementing logical functions [1, 16]. Inspired by the neurons of the biological brain, a lattice based neuron that possesses dendritic structures was developed and is discussed in detail in [13, 14, 17].

Several applications of LNNs have been proposed, due to their high capability of resolving some difficult non-linear problems. LNNs were employed in applications for face and object localization [3, 9], Auto-Associative memories [10, 18, 19], color images retrieval and restoration [20] etc. Furthermore, various models of fuzzy lattice neural networks (FLNN) were studied in [5, 6] and some of their applications in the area of text classification and classification of structured data domains were presented in [7, 8].

Despite the high capabilities of LNNs, training dendritic networks such as Lattice Based Morphological Perceptrons with Dendritic Structure [13], results in creating huge neural networks with a large number of neurons; the size of the trained network sometimes is comparable to the size of the training data.

In this work a new model of LNNs is proposed. In this model each neural dendrite can work on a different orthonormal basis than the other dendrites.

The orthonormal basis of each dendrite is chosen appropriately in order to optimize the performance of the Orthonormal Basis Lattice Neural Network (OB-LNN). OB-LNNs have some useful properties such as automatic compression of the size of the neural network and they show significantly better learning capabilities than the standard basis LNNs. Validation experimental results in synthetic and real datasets are presented and demonstrate superior learning performance of OB-LNNs over SB-LNNs.

The rest of the chapter is organized into the following sections: In Section 3.2, we make a brief review of the LNN model. In Section 3.3, the Orthonormal Basis Lattice Neural Network is presented. This section is divided in two parts. In the first part the model of OB-LNNs is presented. This is followed by an algorithm for training such a neural network. Finally, in Section 3.4, validation experimental results are presented which demonstrate superior learning performance of OB-LNNs over standard basis LNNs.

3.2 Lattice Neural Networks

The primary distinction between traditional neural networks and LNNs is the computation performed by the individual neuron. Traditional neural networks use a multiply accumulate neuron with thresholding over the ring $(\Re, \times, +)$ given by the formula

$$\tau_j(x) = \sum\nolimits_{i=1}^{n} x_i w_{ij} - \theta_j \tag{3.1}$$

where $\tau_j(x)$ is the total input to the j^{th} neuron, x_i are the values of the input neurons connected with the j^{th} neuron, and w_{ij} are their weights. Finally θ_j are bias weights.

In the case of a lattice based neuron, lattice operators $\vee$ (maximum) and $\wedge$ (minimum) are used. These operators and the addition $(+)$ form the rings $(\Re \cup -\infty, \vee, +)$ and $(\Re \cup \infty, \wedge, +)$. The computation performed by a neuron is given by the formula

$$\tau_j(x) = p_j \vee_{i=1}^{n} r_{ij}(x_i + w_{ij}) \tag{3.2}$$

or

$$\tau_j(x) = p_j \wedge_{i=1}^{n} r_{ij}(x_i + w_{ij}) \tag{3.3}$$

where $\tau_j(x)$ is the total input to the j^{th} neuron, x_i are the values of the input neurons connected with the j^{th} neuron, and w_{ij} are their weights. Parameters r_{ij} take +1 or -1 value if the i^{th} input neuron causes excitation or inhibition to the j^{th} neuron. P_j takes also +1 or -1 value if the type of the output response is excitatory or inhibitory. A more detailed presentation of the theory of lattice based morphological neurons and how their networks work can be found in [11, 14].

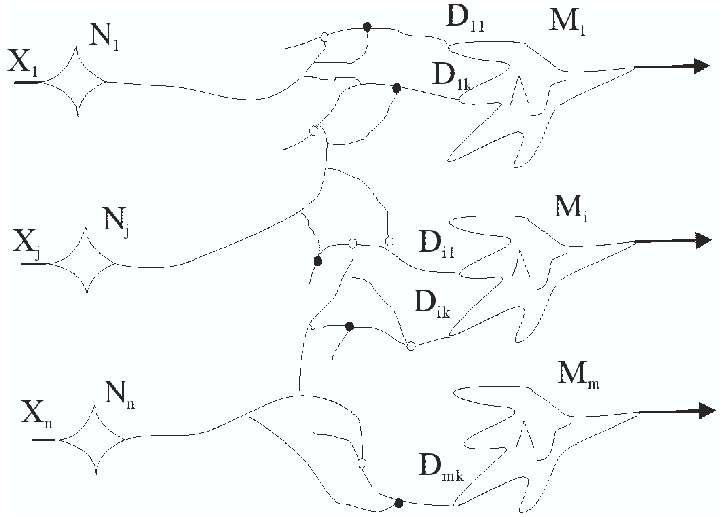

Fig. 3.1. An artificial neural network model which employs lattice based dendritic computations.

Using the above computational framework, a lattice neural network can be constructed using layers of lattice based neurons which are connected to neurons of other layers. Each lattice based neuron consists of dendrites which connect the neuron with the previous layers neurons. Fig. 3.1 shows a lattice neural network with an input layer and a lattice based neuron layer. Each lattice neuron M_j consists of dendrites D_{ij}. The neurons of the input layer are connected to the next layer via the dendrites. The black and white circles denote excitatory and inhibitory connection respectively. Each dendrite can be connected with an input neuron at most twice (with one inhibitory and one excitatory connection).

The computation performed by a single (the k^{th}) dendrite can be expressed using lattice operators by the formula:

$$\tau_k(x) = p_k \wedge_{i \in I} \wedge_{l \in L} (-1)^{1-l}(x_i + w_{ik}^l) \tag{3.4}$$

where I is a subset of 1,...,n which corresponds to the set of all input neurons N_i with terminal fibers that synapse on the k^{th} dendrite of the current lattice based neuron. L is a subset of 1,0 and the other parameters are the same with those used in Eqs. (3.2) and (3.3).

The geometrical interpretation of the computation performed by a dendrite is that every single dendrite defines a hyperbox. The borders of this hyperbox form the decision boundaries in a particular location of the input space. The left part of Fig. 3.2 shows a hyperbox that separates data points of two different classes. Here the input space is the plane of real numbers ($\Re^2$) and the hyperbox is a rectangle. This hyperbox can be defined by a single dendrite via its weight values w_{ij}.

Decision boundaries with more complex shapes, can be formed by using more dendrites. Furthermore boundaries which separate more than two classes can be formed, if more lattice based neurons are employed. The left part of Fig. 3.3 shows an example of decision boundaries with more complex shape, forming the letters ABC. In the middle of the same figure we can see how a group of hyperboxes (2-dimensional boxes in this case) can approximate the decision boundaries. Each box can be defined by a dendrite. Complicated figures require a large number of dendrites, in order to achieve satisfactory approximation of the decision boundaries.

Note that here the word hyperbox has a more general meaning, since some of the bounding planes of the hyperboxes may have infinite length. The word hyperbox as used in this article includes *open* hyperboxes. For example, in $\Re^1$ a half-open interval of form $[\alpha,\infty)$ or $(-\infty,\alpha]$ are half-open *boxes*, while in $\Re^2$ a rectangle as well as the convex area bounded by one, two, or three mutually orthogonal lines are also considered to be hyperboxes. Following similar reasoning, two parallel lines can be also considered as a hyperbox, etc.

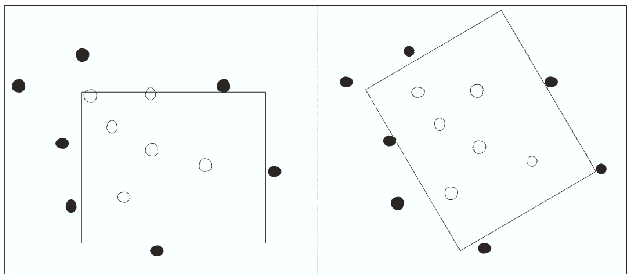

Fig. 3.2. Decision boundaries of a SB-LNN dendrite (left) and a OB-LNN dendrite (right).

An algorithm for training a lattice neural network with dendritic structure can be found in [11] and method for learning LNNs can be found in [17]. A comparison of various training methods for LNNs that employ dendritic computing is presented in [12]. The next section is divided into two parts. In the first part the model of Orthonormal Basis lattice neural network is presented. This is followed by an algorithm for training such a neural network.

3.3 Orthonormal Basis Lattice Neural Networks

In this section, the Orthonormal Basis Lattice Neural Network is presented. This section is divided into two parts. In the first part the model of OB-LNNs is presented. This is followed by an algorithm for training such a neural network.

3.3.1 The OB-LNN model

As it was discussed earlier, the geometrical interpretation of the computation performed by a dendrite is that every single dendrite defines a hyperbox in the space of input values. These hyperboxes are oriented parallel to the Cartesian axis of the space of input values. The left part of Fig. 3.2 presents a decision hyperbox defined by a dendrite. Its edges are parallel to the x and y axis of the input space ($\Re^2$). Due to this constraint about the orientation of the hyperboxes, the decision boundaries formed by lattice neural networks are not smooth and box patterns are annoyingly visible along the boundaries (Fig. 3.3 middle).

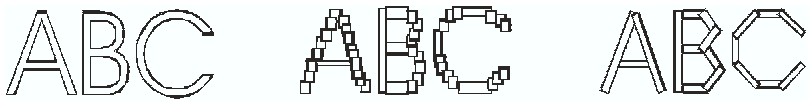

Fig. 3.3. This is an illustration of the decision boundaries that can be formed by a SB-LNN (middle) and an OB-LNN (right). The desired decision boundaries form the ABC letters (left).

To overcome these disadvantages, a new type of dendrite can be defined, which is able to form a hyperbox parallel to an arbitrary orthonormal system. The right part of Fig. 3.2 presents such a hyperbox, whose orientation is no longer parallel to the Cartesian axis of the input space. A neural network consisting of lattice based neurons with such dendrites would be able to produce smoother decision boundaries (Fig. 3.3 right). In Fig. 3.3 the decision boundaries formed by such an Orthonormal Basis Lattice Neural Network are compared with these formed by a Standard Basis LNN. Notice that the number of hyperboxes (thus the number of dendrites as well) required by the OB-LNN is much smaller than the number of those required by a SB-LNN.

Another advantage of the Orthonormal Basis LNNs is that they store information about the local orientation of the classes. This is demonstrated with an example in Fig. 3.4. Suppose that the samples of a class form the shape of the letter A. A neural network with Orthonormal Basis Dendtrites can approximate this shape forming mainly 3 hyperboxes. The orientation of each hyperbox contains information about the local orientation of this class (Fig. 3.4 right). This useful information cannot be obtained by the standard basis LNN that was trained for the same purpose (Fig. 3.4 left). This property is better illustrated in Section 3.4 (Fig. 3.6), where training results of standard and orthonormal basis LNNs are presented.

The computation performed by the k^{th} Orthonormal Basis dendrite can be expressed using lattice operators, changing slightly Eqn. (3.4) as follows

$$\tau_k(\mathbf{X}) = p_k \wedge_{i\in I} \wedge_{l\in L} (-1)^{1-l} \left[(\mathbf{R}_k \mathbf{X})_i + w_{ik}^l \right] \tag{3.5}$$

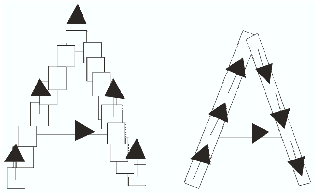

Fig. 3.4. Another advantage of OB-LBNN (right) is that they also store information about the local orientation of the classes.

where $\mathbf{X}$ is the input value vector $(x_1, x_2, ..., x_I)^T$, $\mathbf{R}_k$ is a square matrix whose columns are unit vectors forming an orhonormal basis, and $(\mathbf{R}_k\mathbf{X})_i$ is the i^{th} element of the vector $\mathbf{R}_k\mathbf{X}$. Each dendrite now works in its own orthonormal basis defined by the matrix $\mathbf{R}_k$. The weights of the dendrite act on the elements of vector $\mathbf{R}_k\mathbf{X}$, hence the weights act on the rotated by the orthonormal basis $\mathbf{R}_k$ space.

Note that Standard Basis Lattice Neural Networks are a sub group of the Orthonormal Basis Lattice Neural Networks where the matrix $\mathbf{R}_k$ is the identity matrix. In this case it is obvious $(\mathbf{R}_k\mathbf{X})_i = x_i$, thus Eqn. (3.5) becomes equal to Eqn. (3.4).

3.3.2 Training OB-LNNs

The training of an Orthonormal Basis Lattice Neural Network is based on finding the best possible values for the weights and $\mathbf{R}_k$ matrices. In other words, in order to train an OB-LNN, one must train its dendrites.

The training methods which can be used for training SB-LNNs can also be used for training OB-LNNs with an appropriate modification in order to adopt the fact that each dendrite works on its own orthonormal basis. The training procedure of an orthonormal basis dendrite can be treated as a maximization problem. The quantity that we are trying to maximize is the volume of the hyperbox which is defined by the dendrite. Due to this volume maximization process the OB-LNN training algorithm generally produces more compressed neural networks compared to those in the standard basis case. This property is expressed by lemma 1, which is discussed at the end of this section.

By fixing the matrix $\mathbf{R}_k$, the weights of the dendrite can be estimated by the training procedure described in [11]. The weights define the hyperbox; therefore its volume can be directly calculated from the weights. We can repeat the process by varying appropriately the matrix $\mathbf{R}_k$, until the volume of the hyperbox reaches a maximum.

The matrix $\mathbf{R}_k$ is a rotation matrix, i.e. it rotates the vector $\mathbf{X}$. A variation of this matrix $d\mathbf{R}$ is also a rotation matrix. A new rotation matrix $\mathbf{R'}_k$ can be obtained by multiplying $\mathbf{R}_k$ with matrix $d\mathbf{R}$, $(\mathbf{R'}_k = \mathbf{R}_k d\mathbf{R})$. A variation matrix $d\mathbf{R}$ can be easily constructed by using the following equation:

$$d\mathbf{R} = exp(t\mathbf{S}) \tag{3.6}$$

where $\mathbf{S}$ is a randomly generated skew symmetric matrix, and t is a scalar value. Note that the exponential is the matrix exponential. Note also that the matrix exponential of a skew symmetric matrix is always a rotation matrix. The smaller the absolute value of t is, the smaller the variation, which is caused by the matrix $d\mathbf{R}$, is. A brief review of the properties of matrix exponential and skew-symmetric matrices can be found in the appendix at the end of this article.

By using the above, any cost minimization method can be used in order to minimize the negative of the hyperboxes volume (or to maximize its volume) in steps 2 and 7 of Algorithm 1. Simulated annealing [15] and greedy searching for experiments in 2D are the methods used in the experiments presented in Section 3.4, in order to find the matrix $\mathbf{R}_k$ that gives the maximum hyperboxes volume.

The training algorithm of an OB-LNN morphological perceptron is summarized below. This algorithm is an extension, in the space of OB-LNN, of the training algorithm for morphological perceptron proposed in [11]. The algorithm is presented for the case of 2 classes only, but it can be easily extended to problems with a larger number of classes. A more detailed description of the training algorithm in the case of SB-LNN is presented in [11].

input : N training samples $\mathbf{X}_i$, and N outputs $d_i = 0$ or 1 for class C1 or C2, respectively, $i = 1, ..., N$
output : The number of generated dendrites L, their weights $\mathbf{W}_j$ and their orthonormal basis $\mathbf{R}_j$, $j = 1, ..., L$

Step 1: $L \leftarrow 1$;
Step 2: Find the size $\mathbf{W}$ and orientation $\mathbf{R}$ of the smallest possible hyperbox containing all the samples of C1 ;
Step 3: Assign the result of step 2 to $\mathbf{W}_1$ and $\mathbf{R}_1$;
Step 4: If there are misclassified points of C2, go to step 5 ; else go to step 10 ;
Step 5: Pick arbitrarily a misclassified point ξ of C2 ;
Step 6: $L \leftarrow L + 1$;
Step 7: Find the size $\mathbf{W}$ and orientation $\mathbf{R}$ of the biggest hyperbox that contains ξ, but it does not contain any point of C1 ;
Step 8: Assign the result of step 7 to $\mathbf{W}_L$ and $\mathbf{R}_L$;
Step 9: Go to step 4 ;
Step 10: Terminate the algorithm and report the results ;

Algorithm 1: Training an orthonormal basis lattice neural network

It can be easily shown that the time complexity of this algorithm is equal to $O_{OBLNN} = O_{SBLNN} \times O_{Min}$ where O_{SBLNN} is the time complexity of the training algorithm in the case of SB-LNN and O_{Min} is the time complexity of the minimization process used in steps 2 and 7. Therefore, in order to obtain

improved learning capability we loose in speed performance. The improvement in speed performance will be one of the research topics in our future work. Practically we can use efficiently this algorithm for problems defined in 2 dimensions, e.g. image processing related problems and problems defined in 3 dimensions, e.g. point set processing and 3D volume image processing problems.

More specifically, in 2-dimensional problems the orthonormal basis is defined by a 2×2 rotation matrix $\mathbf{R}$. For the storage of this matrix we need to store only 1 real number, which is either the only lower triangular element of a 2×2 skew-symmetric matrix (see in the Appendix to this chapter), or the rotation angle of the orthonormal system around the origin. In the 3-dimensional case, the orthonormal basis is defined by a 3×3 rotation matrix, which can be stored using only 3 real number storage units. Similarly to the 2D case these 3 numbers are the three lower triangular elements of a 3×3 skew-symmetric matrix $\mathbf{A}$ such that $\mathbf{R} = exp(\mathbf{A})$. Generally in the n-dimensional case, we need $n(n-1)/2$ real number storage units in order to store the orthonormal basis.

The difference between the training algorithm of an orthonormal basis lattice neural network and the training algorithm of a standard basis lattice neural network, is in the processes performed in steps 2 and 7. For the standard basis case these two steps find only the size $\mathbf{W}$ and not the orientation $\mathbf{R}$ of the new dendrite. The standard orthonormal basis (expressed by the identity matrix) is employed for every dendrite. As a result of this difference between the orthonormal basis and the standard basis algorithms, the orthonormal basis algorithm generally produces more compressed neural networks, i.e. with smaller number of dendrites compared to those in the standard basis case. This property is discussed in more details by the following lemma and its proof.

Lemma 1. *The number of dendrites generated by training an orthonormal basis lattice neural network L_O is smaller or equal to the number of dendrites obtained by training on the same dataset a standard basis lattice neural network L_S ($L_O \leq L_S$).*

The proof of the lemma involves understanding of the process performed in step 7 of Algorithm 1. Without loss of generality we will assume that in step 5 the same misclassified point ξ is arbitrarily selected for both orthonormal basis and standard basis lattice neural network training. Let assume that for one particular ξ the SB-LNN algorithm creates a hyperbox of volume A. The OB-LNN algorithm will create a hyperbox with the maximum possible volume, say B. Therefore $A \leq B$, and the equality holds in the case that the standard basis hyperbox happens to have the maximum possible volume. Hence the number of the training points included in volume A must be smaller or equal to those included in volume B. As a consequence the number of iterations performed by the orthonormal basis training algorithm is smaller or equal to those performed by the standard basis training algorithm. This proves the lemma since $L_O \leq L_S$.

3.4 Experimental Results

In this section, validation experimental results are presented which demonstrate superior learning performance of OB-LNNs over SB-LNNs. The experiments were performed using synthetic 2-dimentional datasets and the well known Iris flower dataset.

The first dataset was synthetically generated and it forms two 2D spirals. The points of the two spirals obey the equations

$$[x_1(\theta), y_1(\theta)] = [2\theta cos(\theta)/\pi, 2\theta sin(\theta)/\pi] \tag{3.7}$$

and

$$[x_2(\theta), y_2(\theta)] = [-x_1(\theta), -y_1(\theta)] \tag{3.8}$$

respectively. Several versions of this datasets were generated with a) 130, b) 258, c) 514, d) 770, e) 1026, f) 1538 samples. The largest dataset (2538 samples) was used for the testing dataset. The samples of the smallest dataset (130 samples) are presented in Fig. 3.5. Small circles denote the samples of the one spiral, and small crosses denote the samples of the other one.

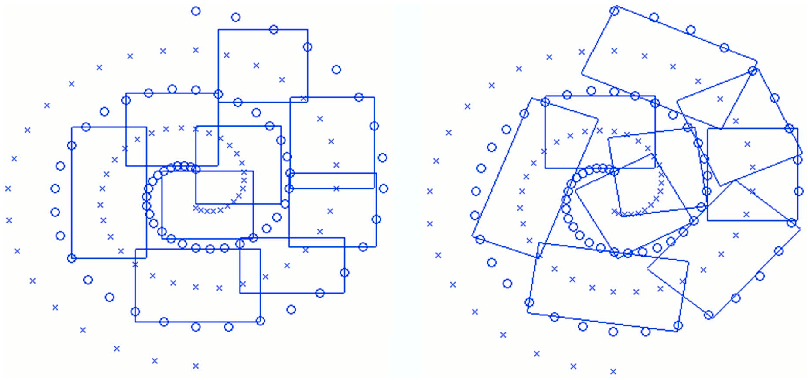

Fig. 3.5. This figure shows some of the hyperboxes formed by a SB-LNN (left) and a OB-LNN(right). The 130 training samples are denoted by circles and crosses and they are forming two spirals. The hyperboxes presented here for comparison, are the 9 smallest of each case.

Two neural networks were trained in the previously described datasets: a) a Standard Basis Lattice Neural Network and b) the proposed Orthonormal Basis Lattice Neural Network. Both networks were lattice based morphological perceptrons with dendritic structure [11] so that their performance could be compared directly. In the case of the OB-LNN perceptron, the dendrites were Orthonormal Basis dendrites, which were trained in order to maximize the volume of the hyperboxes that they formed.

Table 3.1 presents the classification errors of the trained neural networks for different sizes of the training datasets (column 1). The size of the testing dataset for all the experiments was 1538 samples. In all cases, the classification errors made by the OB-LNN are significantly smaller than these made by the SB-LNN. This conclusively demonstrates superior training performance of the proposed neural network over the standard basis lattice neural networks.

Table 3.1. This Table presents classification errors and number of dendrites needed for the training of an OB-LNN and a SB-LNN, using different sizes of training samples (see column 1). The 2^{nd} and 4^{th} columns present the number of dendrites needed for the correct classification of the training samples. The 3^{rd} and 5^{th} columns show the percentage of misclassified samples using always 1538 testing samples. The last column shows the quantity equals to one minus the ratio of the 3^{rd} column over the 5^{th} column.

Training samples	OB-LNN Dendrites	OB-LNN Error	SB-LNN Dendrites	SB-LNN Error	1-Ratio of errors
130	12	14.0 %	14	18.34 %	23.45 %
258	14	6.5 %	17	9.69 %	32.20 %
514	15	3.2 %	19	4.55 %	28.57 %
770	16	2.3 %	19	3.20 %	26.88 %
1026	15	1.5 %	20	2.54 %	40.94 %

Furthermore, the number of the dendrites, which are required to form the decision boundaries between some populations, measures the learning ability of a neural network. Table 3.1 also presents the final number of dendrites required by the lattice neural networks in order to classify correctly all the training samples (columns 2 and 4). In all cases, the number of dendrites trained by the OB-LNN is smaller than the number of dendrites trained by a standard basis LNN. This means that the OB-LNN compresses automatically its size.

Fig. 3.5 shows most of the hyperboxes formed by the dendrites of the Standard basis (left) and the Orthonormal basis LNNs (right). These hyperboxes were formed by the training process using 130 training samples. By observing this figure we can see the differences between the decision boundaries formed by the OB-LNN and those formed by the SB-LNN. The hyperboxes generated by the Orthonormal basis LNN have bigger volume (area in the 2D domain) than those generated by the standard basis LNN. In the case of OB-LNN each hyperbox is rotated appropriately because of the fact that it is working on a different orthonormal basis than the other dendrites.

As it was discussed earlier in Section 3.3, each hyperbox contains information about the local orientation of the classes. This property is illustrated in Fig. 3.6. In this figure the decision boundaries between the two spirals generated by a SB-LNN (left) and an OB-LNN (right) are presented. On each

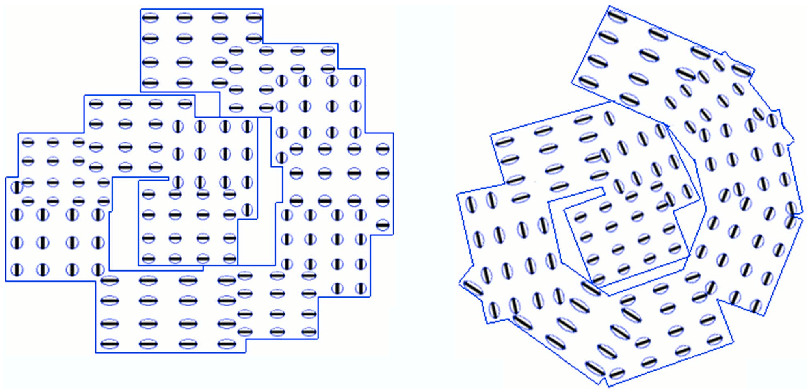

Fig. 3.6. Decision boundaries formed by a SB-LNN (left) and a OB-LNN (right). On each hyperbox several ellipses are plotted. The sizes of the principal axes of each ellipse are proportional to the sizes (length and width) of the relative hyperbox.

hyperbox several ellipses are plotted. The sizes of the principal axes of each ellipse are proportional to the size (length and width) of the relative hyperbox. The dominant axis of each ellipse is also plotted, forming a vector field.

Observing the vector field generated by the SB-LNN (top) and this generated by the OB-LNN (bottom), one can conclude that the hyperboxes of the OB-LNN contain information about the local orientation of the classes. In the case of SB-LNN this property cannot be generally observed.

Another set of experiments was held by using the Iris flower dataset. The Iris Flower dataset is a popular multivariate dataset that was introduced by R.A. Fisher as an example for discriminant analysis. The data reports on four characteristics of the three species of the Iris Flower, sepal length, sepal width, petal length, and petal width. The goal of a discriminant analysis is to produce a simple function that, given the four measurements, will classify a flower correctly.

Several experiments were performed by using randomly different amounts of the Iris flower samples as testing samples (1^{st} column of Table 3.2). The following three neural networks were trained: a) an OB-LNN perceptron, b) a SB-LNN perceptron and c) a multilayer perceptron (MLP) with one hidden layer. Several different architectures were used for the MLP with 1 hidden layer, and the results of Table 3.2 are the best obtained. The whole Iris data set were used as the testing data set. The classification errors of the three neural networks are presented in Table 3.2. In all cases, the classification errors made by the OB-LNN are significantly smaller.

Finally, another experiment was also held in order to compare the sizes of the trained neural networks. A synthetic dataset was generated forming two classes; one within an ellipse (Fig. 3.7 left) and the other one outside of it. The sample points of the two classes were picked up randomly using uniform distribution.

Table 3.2. This Table presents classification errors of the three neural networks for different amounts of training samples.

Training samples	OB-LNN	SB-LNN	Perceptron
50 %	8.33 %	10.67 %	13.33 %
60 %	6.41 %	10.00 %	12.21 %
70 %	4.20 %	6.33 %	7.34 %
80 %	3.12 %	3.67 %	6.54 %
90 %	2.01 %	3.33 %	6.38 %
100 %	0 %	0 %	3.67 %

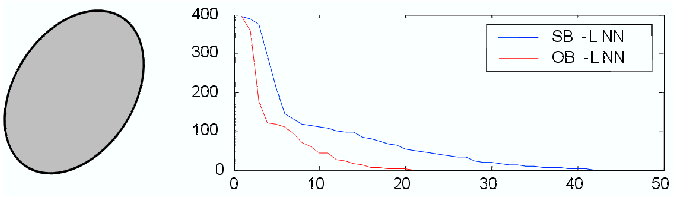

Fig. 3.7. Left: This dataset forms an ellipse. Right: Plot of the misclassified points over the number of dendrites required by a OB-LNN and a SB-LNN during the training process.

The same two lattice neural networks with dendritic structure were used: a) an OB-LNN perceptron and b) a SB-LNN perceptron. The right plate of Fig. 3.7 shows the plot of the number of misclassified sample points over the number of dendrites generated by the two neural networks during the training process. The final number of dendrites required by the OB-LNN in order to classify correctly all the training samples is significantly smaller than the number of dendrites trained by a standard basis LNN. This also conclusively demonstrates superior learning performance of OB-LNNs over SB-LNNs.

3.5 Conclusion

A novel model, namely Orthonormal Basis Lattice Neural Network, was presented. Comparisons of the proposed model with the standard basis model of Lattice neural networks were shown. Validation experimental results were also presented demonstrating the advantages of the proposed model. Our future work will be focused on applying this model in applications for face and object localization, Auto-Associative memories and color images retrieval and restoration, in which areas the standard basis lattice neural networks have been applied [3, 9, 10, 18, 19, 20]. Extension of the proposed model in the area of Fuzzy Lattice Neurocomputing [5, 6] and improvement in speed performance will also be some of our future research topics.

References

1. Eccles J (1977) The Understanding of the Brain. McGraw-Hill, New York
2. Gallier J, Xu D (2003) Computing exponentials of skew-symmetric matrices and logarithms of orthogonal matrices. Intl J of Robotics and Autom 18:10–20
3. Grana M, and Raducanu B (2001) Some applications of morphological neural networks. In: Proc Intl Joint Conf Neural Networks 4:2518–2523
4. Horn RA, Johnson CR (1991) Topics in Matrix Analysis. Cambridge University Press
5. Kaburlasos V, Petridis V (2000) Fuzzy lattice neurocomputing (FLN) models. Neural Networks 13(10):1145–1170
6. Petridis V, Kaburlasos V (1998) Fuzzy lattice neural network (FLNN): a hybrid model for learning. IEEE Transactions on Neural Networks 9(5):877–890
7. Petridis V, Kaburlasos V (2001) Clustering and classification in structured data domains using fuzzy lattice neurocomputing (FLN). IEEE Transactions on Knowledge and Data Engineering 13(2):245–260
8. Petridis V, Kaburlasos V, Fragkou P, Kehagias A (2001) Text classification using the σ-FLNMAP neural network. In: Proc Intl Joint Conference on Neural Networks 2:1362–1367
9. Raducanu B, Grana M (2001) Morphological neural networks for vision based self-localization. In: Proc IEEE Intl Conference on Robotics and Automation 2:2059–2064
10. Ritter GX, Iancu L (2004) A morphological auto-associative memory based on dendritic computing. In: Proc IEEE Intl Joint Conference on Neural Network 2:915–920
11. Ritter GX, Iancu L, Urcid G (2003) Morphological perceptrons with dendritic structure. In: Proc IEEE Intl Conference on Fuzzy Systems 2:1296–1301
12. Ritter GX, Schmalz M (2006) Learning in lattice neural networks that employ dendritic computing. In: Proc IEEE World Congress on Computational Intelligence pp 7–13
13. Ritter GX, Sussner P (1996) An introduction to morphological neural networks. In: Proc Intl Conference on Pattern Recognition 4:709–717
14. Ritter GX, Urcid G (2007) Learning in lattice neural networks that employ dendritic computing. This volume, chapter 2
15. Saul A, Teukolsky S, Flannery B, Press W, Vetterling B (2003) Numerical Recipes Example Book (C++). Cambridge University Press
16. Segev I (1988) Dendritic processing. In: Arbib MA (ed) The Handbook of Brain Theory and Neural Networks pp 282–289. MIT press
17. Sussner P (1998) Morphological perceptron learning. In: Proc IEEE ISIC/CIRA/ISAS Joint Conference pp 477–482
18. Sussner P (2004) Binary autoassociative morphological memories derived from the kernel method and the dual kernel method. In: Proc Intl Joint Conference on Neural Networks 1:236–241
19. Sussner P (2004) A fuzzy autoassociative morphological memory. In: Proc Intl Joint Conference on Neural Networks 1:326–331
20. Yun Z, Ling Z, Yimin Y (2004) Using multi-layer morphological neural network for color images retrieval. In: Proc World Congress on Intelligent Control and Automation 5:4117–4119

Chapter 3 Appendix

In this section we review briefly some of the mathematical preliminaries related to the methods presented in this article.

The matrix exponential of a square $n \times n$ matrix $\mathbf{A}$ is defined as

$$exp(\mathbf{A}) = \sum_{i=0}^{\infty} \frac{\mathbf{A}^i}{i!} = \mathbf{I} + \mathbf{A} + \frac{\mathbf{A}^2}{2} + \frac{\mathbf{A}^3}{6} + ... \tag{3.9}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix. The summation of the infinite terms of Eqn. (3.9) does converge and the obtained result is a $n \times n$ positive definite matrix. Similarly to the matrix exponential we can define the matrix logarithm as the inverse operation. The matrix logarithm of a $n \times n$ positive definite real valued matrix $\mathbf{A}$ gives a $n \times n$ real valued matrix. Both exp and log matrix operations are invariant to rotations, i.e. for every $n \times n$ orthogonal matrix $\mathbf{v}$ we have $exp(\mathbf{v}\mathbf{A}\mathbf{v}^T) = \mathbf{v}exp(\mathbf{A})\mathbf{v}^T$ and $log(\mathbf{v}\mathbf{A}\mathbf{v}^T) = \mathbf{v}log(\mathbf{A})\mathbf{v}^T$.

A skew-symmetric matrix $\mathbf{A}$ is a $n \times n$ matrix whose negative is also the transpose of itself (i.e. $\mathbf{A}^T = -\mathbf{A}$). A skew-symmetric matrix has all its diagonal elements zero, and the rest of the elements satisfy the property $A_{i,j} = -A_{j,i}$.

The matrix exponential of a skew-symmetric matrix is an orthogonal matrix $\mathbf{R} = exp(\mathbf{A})$, where $\mathbf{R}$ is an orthogonal matrix and it is also a positive definite matrix because it is expressed as the matrix exponential of a matrix. The zero $n \times n$ matrix is an example of a skew-symmetric matrix. By evaluating Eqn. (3.9) it turns out that the matrix exponential of the zero matrix is the $n \times n$ identity matrix (i.e. $\mathbf{I} = exp(\mathbf{0})$), which is also an orthogonal matrix. The $n \times n$ identity matrix $\mathbf{I}$ can be seen as the zero rotation matrix of the n-dimensional space. Generally, the matrix operation $exp(\lambda\mathbf{A})$, where λ is a scalar which is close to zero, produces an orthogonal matrix which is close to the identity and rotates slightly the n-dimensional space.

$$\lim_{\lambda \to 0} exp(\lambda\mathbf{A}) = \mathbf{I} \tag{3.10}$$

A random small rotation of the n-dimensional space can be generated from Eqn. (3.10) by using a randomly generated skew-symmetric matrix $\mathbf{A}$ and a scalar λ which is close to zero. A random skew-symmetric matrix can be generated by using the multivariate Gaussian distribution for the lower triangular elements of the matrix. If we set the rest of the elements (diagonal and upper triangular) to be zero, we can obtain a randomly generated skew-symmetric matrix by subtracting the transpose of this lower triangular matrix from itself.

Computing exponential of a $n \times n$ skew-symmetric matrix has time complexity $O(n^3)$. A detailed discussion on computing exponentials of skew-symmetric matrices and logarithms of orthogonal matrices was presented in [2]. Finally, a more detailed study on the matrix exponential, skew-symmetric matrices and the related theory can be found in [4].