6

Neural Network Architectures

6.1 Introduction **6**-1
6.2 Special Easy-to-Train Neural Network Architectures **6**-1
Polynomial Networks • Functional Link Networks • Sarajedini and Hecht-Nielsen Network • Feedforward Version of the Counterpropagation Network • Learning Vector Quantization • WTA Architecture • Cascade Correlation Architecture • Radial Basis Function Networks • Implementation of RBF Networks with Sigmoidal Neurons • Networks for Solution of Parity-*N* Problems • Pulse-Coded Neural Networks
6.3 Comparison of Neural Network Topologies **6**-13
6.4 Recurrent Neural Networks **6**-14
Hopfield Network • Autoassociative Memory • BAM—Bidirectional Autoassociative Memories
References **6**-16

Bogdan M. Wilamowski
Auburn University

6.1 Introduction

Different neural network architectures are widely described in the literature [W89,Z95,W96,WJK99, H99,WB01,W07]. The feedforward neural networks allow only for one directional signal flow. Furthermore, most of the feedforward neural networks are organized in layers. An example of the three layer feedforward neural network is shown in Figure 6.1. This network consists of three input nodes: two *hidden layers* and an output layer. Typical activation functions are shown in Figure 6.2. These continuous activation functions allow for the gradient-based training of multilayer networks.

6.2 Special Easy-to-Train Neural Network Architectures

Training of multilayer neural networks is difficult. It is much easier to train a single neuron or a single layer of neurons. Therefore, several concepts of neural network architectures were developed where only one neuron can be trained at a time. There are also neural network architectures where training is not needed [HN87,W02]. This chapter reviews various easy-to-train architectures. Also, it will be shown that abilities to recognize patterns strongly depend on the used architectures.

6.2.1 Polynomial Networks

Using nonlinear terms with initially determined functions, the actual number of inputs supplied to the one layer neural network is increased. In the simplest case, nonlinear elements are higher order polynomial terms of input patterns.

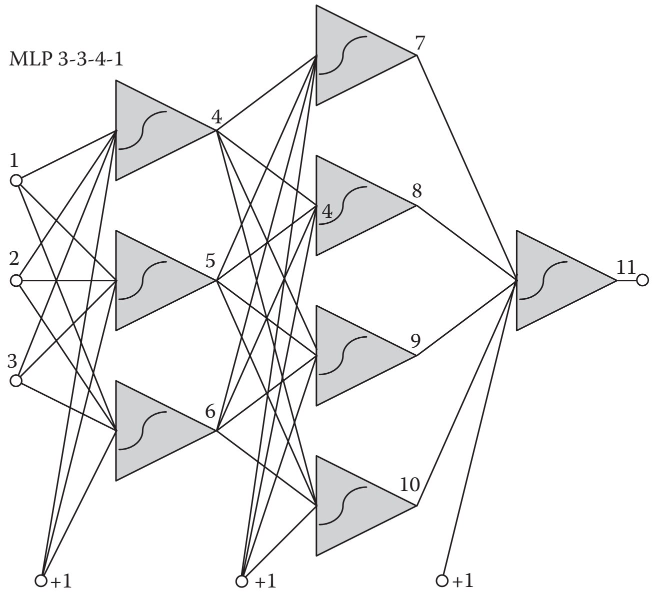


FIGURE 6.1 MLP type architecture 3-3-4-1 (without connections across layers).

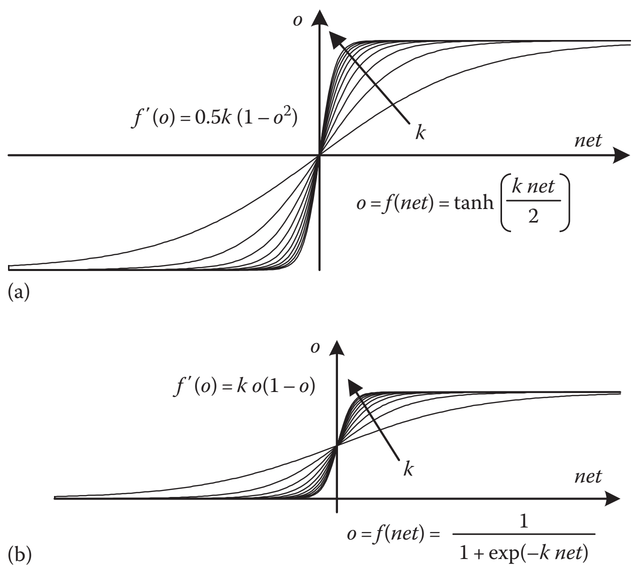


FIGURE 6.2 Typical activation functions: (a) bipolar and (b) unipolar.

The learning procedure for one layer is easy and fast. Figure 6.3 shows an *XOR* problem solved using functional link networks. Figure 6.4 shows a single trainable layer neural network with nonlinear polynomial terms. The learning procedure for one layer is easy and fast.

Note that when the polynomial networks have their limitations, they cannot handle networks with many inputs because the number of polynomial terms may grow exponentially.

6.2.2 Functional Link Networks

One-layer neural networks are relatively easy to train, but these networks can solve only linearly separated problems. One possible solution for nonlinear problems was elaborated by Pao [P89] using the

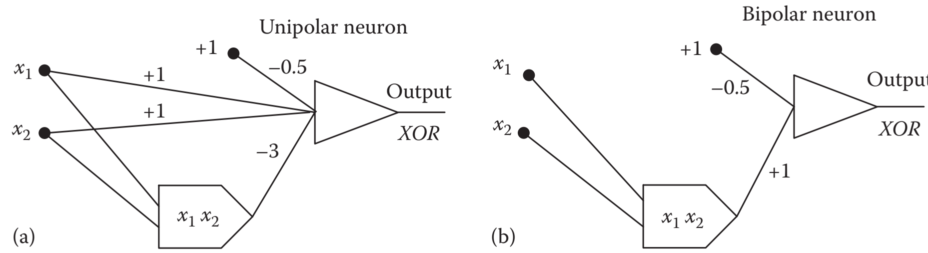


FIGURE 6.3 Polynomial networks for solution of the *XOR* problem: (a) using unipolar signals and (b) using bipolar signals.

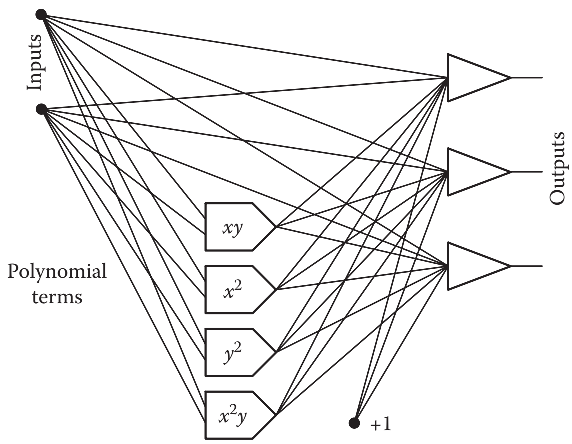


FIGURE 6.4 One layer neural network with nonlinear polynomial terms.

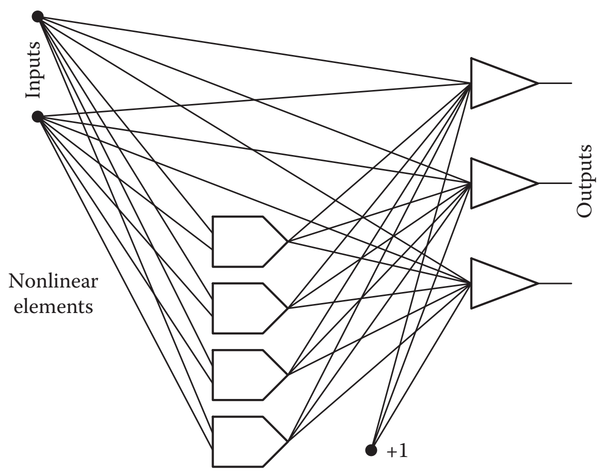
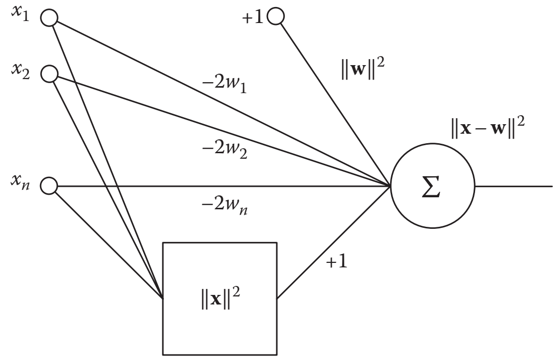


FIGURE 6.5 One layer neural network with arbitrary nonlinear terms.

functional link network shown in Figure 6.5. Note that the functional link network can be treated as a one-layer network, where additional input data are generated off-line using nonlinear transformations.

Note that, when the functional link approach is used, this difficult problem becomes a trivial one. The problem with the functional link network is that proper selection of nonlinear elements is not an easy task. However, in many practical cases it is not difficult to predict what kind of transformation of input data may linearize the problem, so the functional link approach can be used.



$$\|\mathbf{x}-\mathbf{w}\|^2 = \mathbf{x}^T\mathbf{x} + \mathbf{w}^T\mathbf{w} - 2\mathbf{x}^T\mathbf{w} = \|\mathbf{x}\|^2 + \|\mathbf{w}\|^2 - 2net$$

FIGURE 6.6 Sarajedini and Hecht-Nielsen neural network.

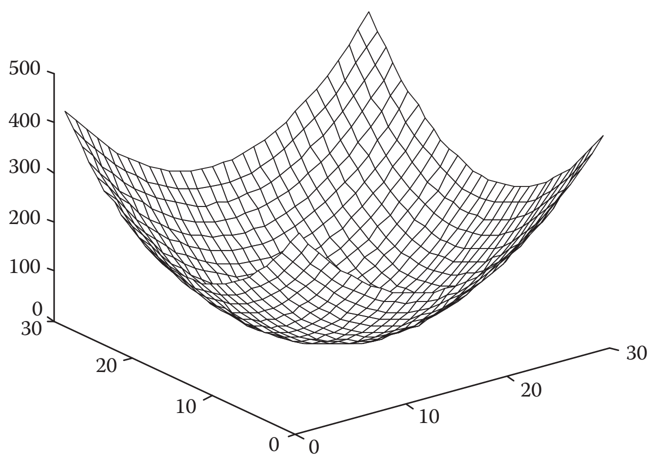


FIGURE 6.7 Output of the Sarajedini and Hecht-Nielsen network is proportional to the square of Euclidean distance.

6.2.3 Sarajedini and Hecht-Nielsen Network

Figure 6.6 shows a neural network which can calculate the Euclidean distance between two vectors **x** and **w**. In this powerful network, one may set weights to the desired point **w** in a multidimensional space and the network will calculate the Euclidean distance for any new pattern on the input. The difficult task is the calculate $\|\mathbf{x}\|^2$, but it can be done off-line for all incoming patterns. A sample output for a two-dimensional case is shown in Figure 6.7.

6.2.4 Feedforward Version of the Counterpropagation Network

The counterpropagation network was originally proposed by Hecht-Nilsen [HN87]. In this chapter, a modified feedforward version as described by Zurada [Z92] is discussed. This network, which is shown in Figure 6.8, requires the number of hidden neurons to be equal to the number of input patterns, or, more exactly, to the number of input clusters.

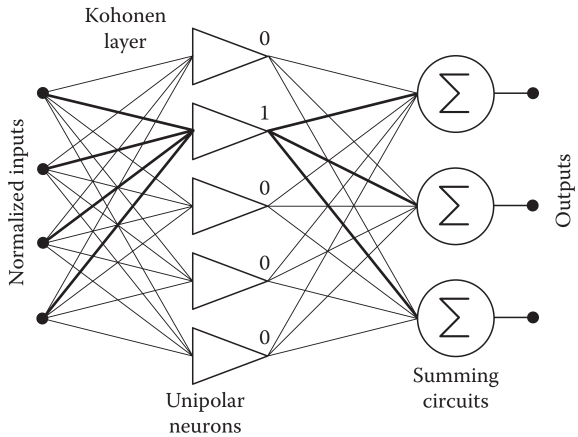


FIGURE 6.8 Counterpropagation network.

When binary input patterns are considered, then the input weights must be exactly equal to the input patterns. In this case,

$$net = \mathbf{x}^t \mathbf{w} = \left(n - 2HD(\mathbf{x}, \mathbf{w})\right) \tag{6.1}$$

where
- n is the number of inputs
- $\mathbf{w}$ are weights
- $\mathbf{x}$ is the input vector
- $HD(\mathbf{x}, \mathbf{w})$ is the Hamming distance between input pattern and weights

In order that a neuron in the input layer is reacting just for the stored pattern, the threshold value for this neuron should be

$$w_{n+1} = -(n-1) \tag{6.2}$$

If it is required that the neuron must react also for similar patterns, then the threshold should be set to $w_{n+1} = -(n - (1 + HD))$, where HD is the Hamming distance defining the range of similarity. Since for a given input pattern, only one neuron in the first layer may have the value of one and the remaining neurons have zero values, the weights in the output layer are equal to the required output pattern.

The network, with unipolar activation functions in the first layer, works as a look-up table. When the linear activation function (or no activation function at all) is used in the second layer, then the network also can be considered as an analog memory (Figure 6.9) [W03,WJ96].

The counterpropagation network is very easy to design. The number of neurons in the hidden layer should be equal to the number of patterns (clusters). The weights in the input layer should be equal to the input patterns and, the weights in the output layer should be equal to the output patterns. This simple network can be used for rapid prototyping. The counterpropagation network usually has more hidden neurons than required.

6.2.5 Learning Vector Quantization

At learning vector quantization (LVQ) network (Figure 6.10), the first layer detects subclasses. The second layer combines subclasses into a single class. First layer computes Euclidean distances between input pattern and stored patterns. Winning "neuron" is with the minimum distance.

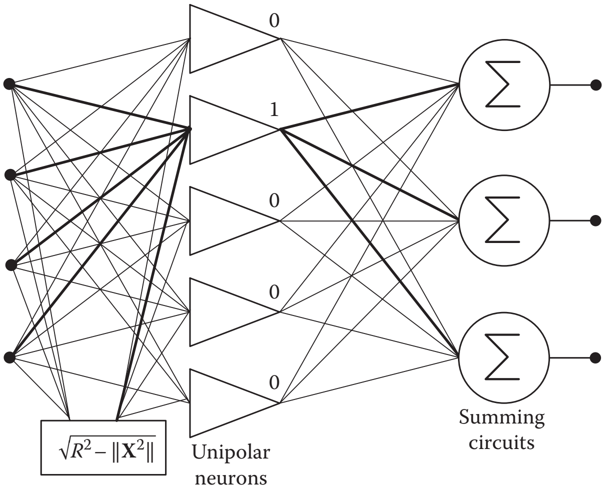


FIGURE 6.9 Counterpropagation network used as analog memory with analog address.

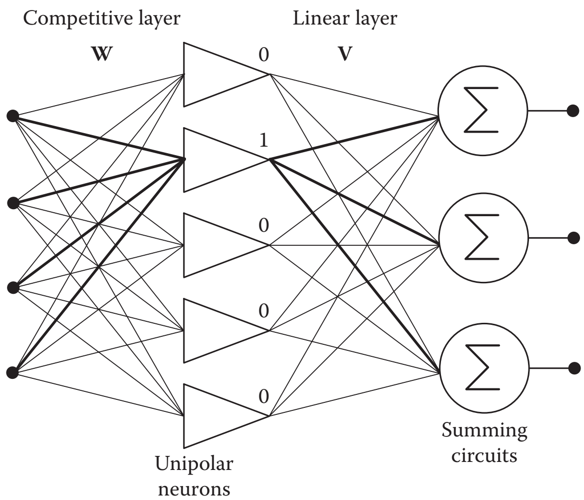


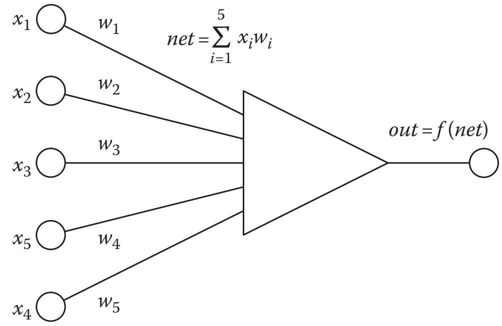
FIGURE 6.10 Learning vector quantization.

6.2.6 WTA Architecture

The winner-take-all (WTA) network was proposed by Kohonen [K88]. This is basically a one-layer network used in the unsupervised training algorithm to extract a statistical property of the input data. At the first step, all input data is normalized so that the length of each input vector is the same, and usually equal to unity. The activation functions of neurons are unipolar and continuous. The learning process starts with a weight initialization to small random values.

Let us consider a neuron shown in Figure 6.11. If inputs are binaries, for example $\mathbf{X}=[1, -1, 1, -1, -1]$, then the maximum value of *net*

$$net = \sum_{i=1}^{5} x_i w_i = \mathbf{X}\mathbf{W}^T \tag{6.3}$$



AQ1

FIGURE 6.11 Neuron as the Hamming distance classifier.

is when weights are identical to the input pattern **W**=[1, –1, 1, –1, –1]. The Euclidean distance between weight vector **W** and input vector **X** is

$$\|\mathbf{W}-\mathbf{X}\| = \sqrt{(w_1 - x_1)^2 + (w_2 - x_2)^2 + \cdots + (w_n - x_n)^2} \tag{6.4}$$

AQ2

$$\|\mathbf{W}-\mathbf{X}\| = \sqrt{\sum_{i=1}^{n} (w_i - x_i)^2} \tag{6.5}$$

$$\|\mathbf{W}-\mathbf{X}\| = \sqrt{\mathbf{W}\mathbf{W}^T - 2\mathbf{W}\mathbf{X}^T + \mathbf{X}\mathbf{X}^T} \tag{6.6}$$

When the lengths of both the weight and input vectors are normalized to value of 1

$$\|\mathbf{X}\| = 1 \quad \text{and} \quad \|\mathbf{W}\| = 1 \tag{6.7}$$

then the equation simplifies to

$$\|\mathbf{W}-\mathbf{X}\| = \sqrt{2 - 2\mathbf{W}\mathbf{X}^T} \tag{6.8}$$

Please notice that the maximum value of net value *net* = 1 is when **W** and **X** are identical.

Kohonen WTA networks have some problems:

1. Important information about length of the vector is lost during the normalization process
2. Clustering depends on
   a. Order of patterns applied
   b. Number of initial neurons
   c. Initial weights

6.2.7 Cascade Correlation Architecture

The cascade correlation architecture (Figure 6.12) was proposed by Fahlman and Lebiere [FL90]. The process of network building starts with a one-layer neural network and hidden neurons are added as needed.

In each training step, the new hidden neuron is added and its weights are adjusted to maximize the magnitude of the correlation between the new hidden neuron output and the residual error signal on

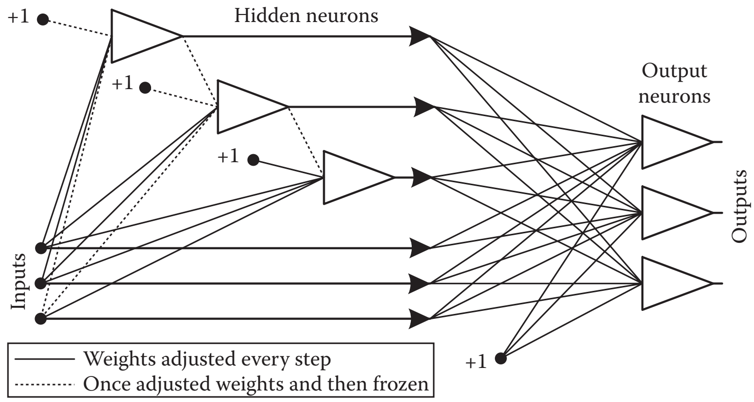


FIGURE 6.12 Cascade correlation architecture.

the network output that we are trying to eliminate. The correlation parameter S defined in the following equation must be maximized:

$$S = \sum_{o=1}^{O} \left| \sum_{p=1}^{P} (V_p - \bar{V})(E_{po} - \overline{E_o}) \right| \tag{6.9}$$

where
- O is the number of network outputs
- P is the number of training patterns
- V_p is output on the new hidden neuron
- E_{po} is the error on the network output

By finding the gradient, $\Delta S/\Delta w_i$, the weight adjustment for the new neuron can be found as

$$\Delta w_i = \sum_{o=1}^{O} \sum_{p=1}^{P} \sigma_o \,(E_{po} - \overline{E_o}) f_{p'} x_{ip} \tag{6.10}$$

The output neurons are trained using the delta (backpropagation) algorithm. Each hidden neuron is trained just once and then its weights are frozen. The network learning and building process is completed when satisfactory results are obtained.

6.2.8 Radial Basis Function Networks

The structure of the radial basis function (RBF) network is shown in Figure 6.13. This type of network usually has only one hidden layer with special "neurons". Each of these "neurons" responds only to the inputs signals close to the stored pattern.

The output signal h_i of the ith hidden "neuron" is computed using formula:

$$h_i = \exp\left(-\frac{\left\| \mathbf{x} - \mathbf{s}_i \right\|^2}{2\sigma^2} \right) \tag{6.11}$$

Note that the behavior of this "neuron" significantly differs from the biological neuron. In this "neuron", excitation is not a function of the weighted sum of the input signals. Instead, the distance between the input and stored pattern is computed. If this distance is zero, then the "neuron" responds

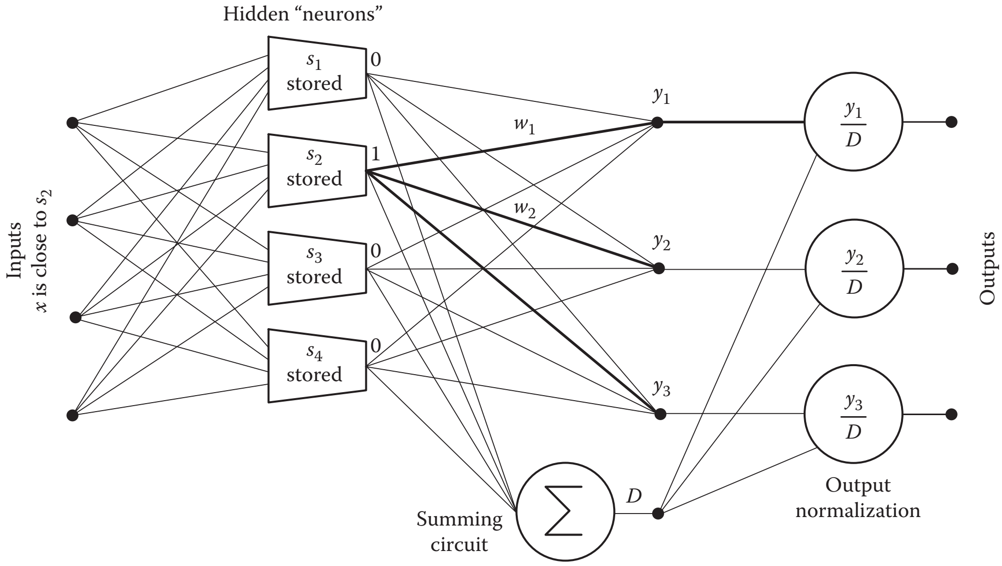


FIGURE 6.13 Radial basis function networks.

with a maximum output magnitude equal to one. This "neuron" is capable of recognizing certain patterns and generating output signals being functions of a similarity.

6.2.9 Implementation of RBF Networks with Sigmoidal Neurons

The network shown in Figure 6.14 has similar property (and power) like RBF networks, but it uses only traditional neurons with sigmoidal activation functions [WJ96]. By augmenting the input space to another dimension the traditional neural network will perform as a RBF network. Please notice that this additional transformation can be made by another neural network. As it is shown in Figure 6.15, 2 first neurons are creating an additional dimension and then simple 8 neurons in one layer feedforward network can solve the two spiral problem. Without this transformation, about 35 neurons are required to solve the same problem with neural network with one hidden layer.

6.2.10 Networks for Solution of Parity-*N* Problems

The most common test benches for neural networks are parity-*N* problems, which are considered to be the most difficult benchmark for neural network training. The simplest parity-2 problem is also known

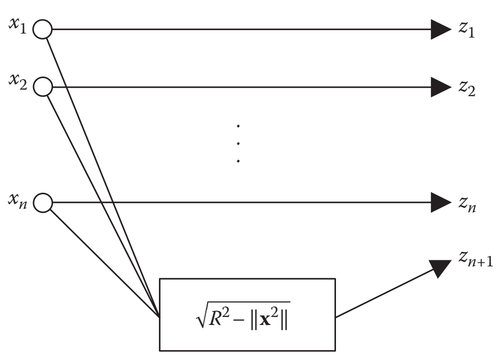


FIGURE 6.14 Transformation, which is required to give a traditional neuron the RBF properties.

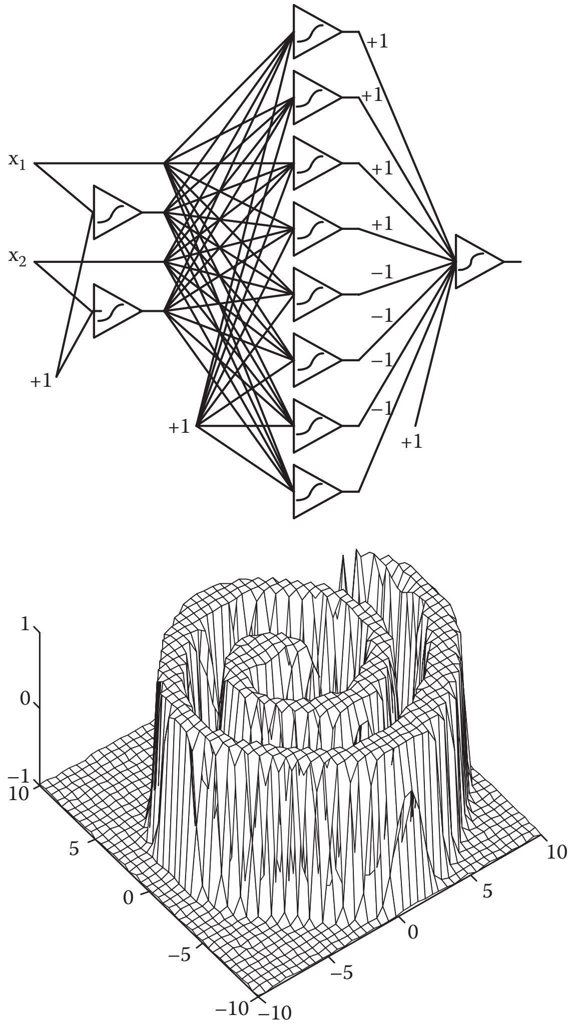
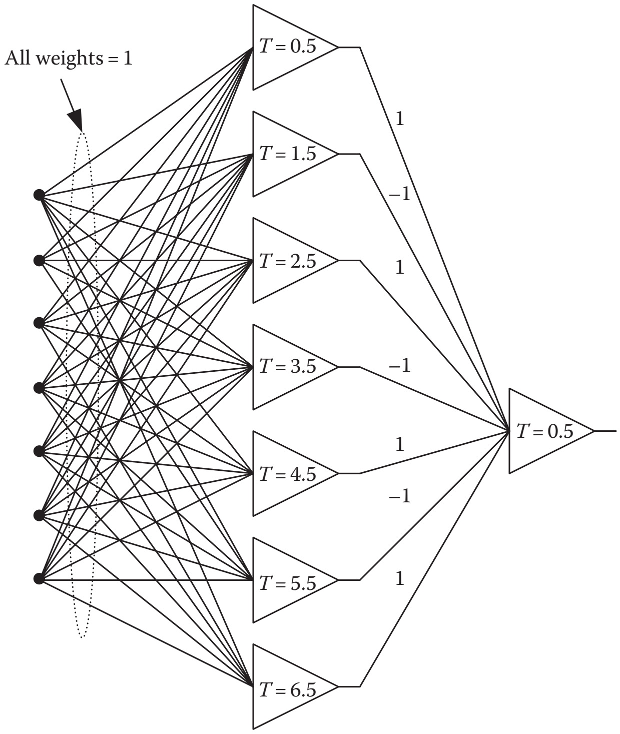


FIGURE 6.15 Solution of two spiral problem using transformation from Figure 6.14 implemented on two additional neurons.

as the *XOR* problem. The larger the *N*, the more difficult it is to solve. Even though parity-*N* problems are very complicated, it is possible to analytically design neural networks to solve them [WHM03,W09]. Let us design neural networks for the parity-7 problem using different neural network architectures with unipolar neurons.

Figure 6.16 shows the multilayer perceptron (MLP) architecture with one hidden layer. In order to properly classify patterns in parity-*N* problems, the location of zeros and ones in the input patterns are not relevant, but it is important how many ones are in the patterns. Therefore, one may assume identical weights equal +1 connected to all inputs. Depending on the number of ones in the pattern, the net values of neurons in the hidden layer are calculated as a sum of inputs times weights. The results may vary from 0 to 7 and will be equal to the number of ones in an input pattern. In order to separate these eight possible cases, we need seven neurons in the hidden layer with thresholds equal to 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, and 6.5. Let us assign positive (+1) and negative (–1) weights to outputs of consecutive neurons starting with +1. One may notice that the net value of the output neuron will be zero for patterns with an odd number of ones and will be one with an even number of ones.



Number of ones in a pattern			0	1	2	3	4	5	6	7
net 1 through *net* 7 (from inputs only)			0	1	2	3	4	5	6	7
out1	$T = 0.5$	$w_1 = 1$	0	1	1	1	1	1	1	1
out2	$T = 1.5$	$w_2 = -1$	0	0	1	1	1	1	1	1
out3	$T = 2.5$	$w_3 = 1$	0	0	0	1	1	1	1	1
out4	$T = 3.5$	$w_4 = -1$	0	0	0	0	1	1	1	1
out5	$T = 4.5$	$w_5 = 1$	0	0	0	0	0	1	1	1
out6	$T = 5.5$	$w_6 = -1$	0	0	0	0	0	0	1	1
out7	$T = 6.5$	$w_7 = 1$	0	0	0	0	0	0	0	1
net 8 = *net* 1 + $\sum_{i=1}^{7} w_i * out_i$			0	1	0	1	0	1	0	1
out8 (of output neuron) $T = 0.5$			0	1	0	1	0	1	0	1

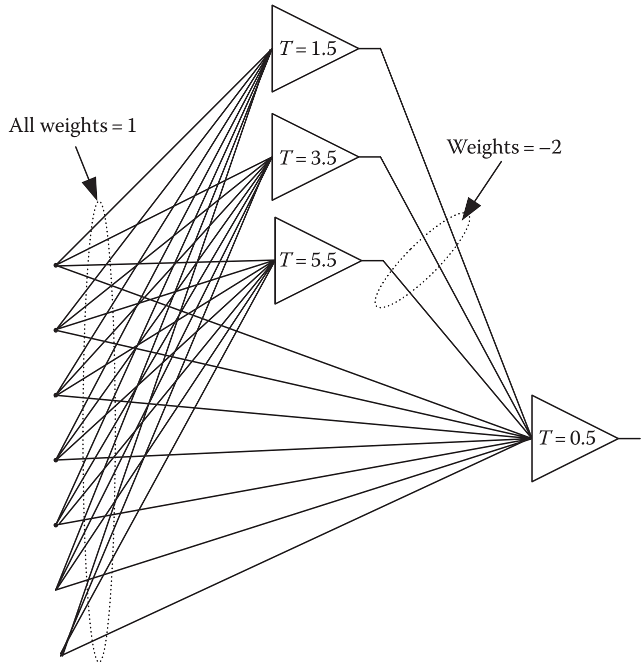
FIGURE 6.16 MLP architecture for parity-7 problem. The computation process of the network is shown in the table.

The threshold of +0.5 of the last neuron will just reinforce the same values on the output. The signal flow for this network is shown in the table of Figure 6.16.

In summary, for the case of a MLP neural network the number of neurons in the hidden layer is equal to $N = 7$ and total number of neurons is 8. For other parity-N problems and MLP architecture:

$$\text{Number of neurons} = N + 1 \tag{6.12}$$

Figure 6.17 shows a solution with bridged multilayer perceptron (BMLP) with connections across layers. With this approach the neural network can be significantly simplified. Only 3 neurons are needed in the hidden layer with thresholds equal to 1.5, 3.5, and 5.5. In this case, all weights associated with outputs of hidden neurons must be equal to –2 while all remaining weights in the network are equal to +1.



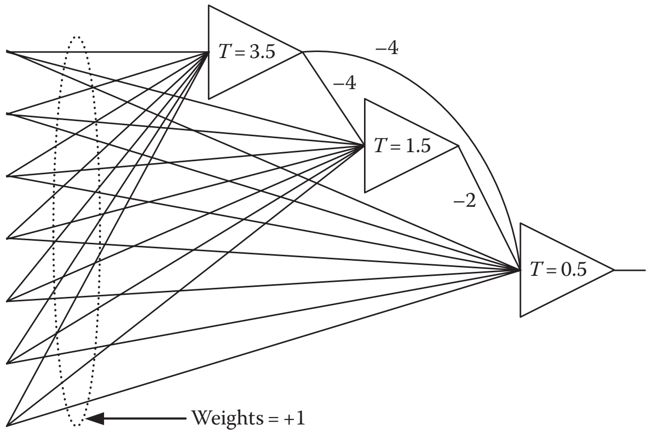
Number of ones in a pattern		0	1	2	3	4	5	6	7
*net*1 (from inputs only)		0	1	2	3	4	5	6	7
out1	$T = 1.5$	0	0	1	1	1	1	1	1
out2	$T = 3.5$	0	0	0	0	1	1	1	1
out3	$T = 5.5$	0	0	0	0	0	0	1	1
*net*1 = *ne*l – 2*(out1 + out2 + out3)		0	1	0	1	0	1	0	1
out4 (of output neuron) $T = 0.5$		0	1	0	1	0	1	0	1

FIGURE 6.17 BMLP architecture with one hidden layer for parity-7 problem. The computation process of the network is shown in the table.

Signal flow in this BMLP network is shown in the table in Figure 6.17. With bridged connections across layers the number of hidden neurons was reduced to $(N - 1)/2 = 3$ and the total number of neurons is 4. For other parity-N problems and BMLP architecture:

$$\text{Number of neurons} = \begin{cases} \dfrac{N-1}{2} + 1 & \text{for odd parity} \\ \dfrac{N}{2} + 1 & \text{for even parity} \end{cases} \tag{6.13}$$

Figure 6.18 shows a solution for the fully connected cascade (FCC) architecture for the same parity-7 problem. In this case, only 3 neurons are needed with thresholds 3.5, 1.5, and 0.5. The first neuron with threshold 3.5 is inactive (out = 0) if the number of ones in an input pattern is less than 4. If the number of ones in an input pattern is 4 or more then the first neuron becomes active and with −4 weights attached to its output it subtracts −4 from the nets of neurons 2 and 3. Instead of [0 1 2 3 4 5 6 7] these neurons will see [0 1 2 3 0 1 2 3]. The second neuron with a threshold of 1.5 and the −2 weight associated with its



		0	1	2	3	4	5	6	7
Number of ones in a pattern		0	1	2	3	4	5	6	7
*net*1 (from inputs only)		0	1	2	3	4	5	6	7
out1 (of first neuron)	*T* = 3.5	0	0	0	0	1	1	1	1
*net*2 = *net*1 – 4*out1		0	1	2	3	0	1	2	3
out2 (of second neuron)	*T* = 1.5	0	0	1	1	0	0	1	1
*net*3 = *net*2 – 2*out2		0	1	0	1	0	1	0	1
out3 (of output neuron)	*T* = 0.5	0	1	0	1	0	1	0	1

FIGURE 6.18 FCC architecture for parity-7 problem. The computation process of the network is shown in the table.

output works in such a way that the last neuron will see [0 1 0 1 0 1 0 1] instead of [0 1 2 3 0 1 2 3]. For other parity-*N* problems and FCC architecture:

$$\text{Number of neurons} = \left\lceil \log 2(N+1) \right\rceil \tag{6.14}$$

6.2.11 Pulse-Coded Neural Networks

Commonly used artificial neurons behave very differently than biological neurons. In biological neurons, information is sent in a form of pulse trains [WJPM96]. As a result, additional phenomena such as pulse synchronization play an important role and the pulse coded neural networks are much more powerful than traditional artificial neurons. Then can be used very efficiently for example for image filtration [WPJ96]. However, their hardware implementation is much more difficult [OW99,WJK99].

6.3 Comparison of Neural Network Topologies

With the design process, as described in section II, it is possible to design neural networks to arbitrarily large parity problems using MLP, BMLP, and FCC architectures. Table I shows comparisons of minimum number of neurons required for these three architectures and various parity-N problems.

As one can see from Table 6.1 and Figures 6.16 through 6.18, the MLP architectures are the least efficient parity-N application. For small parity problems, BMLP and FCC architectures give similar results. For larger parity problems, the FCC architecture has a significant advantage, and this is mostly due to more layers used. With more layers one can also expect better results in BMLP, too. These more powerful neural network architectures require more advanced software to train them [WCKD07,WCKD08,WH10]. Most of the neural network software available in the market may train only MLP networks [DB04,HW09].

TABLE 6.1 Minimum Number of Neurons Required for Various Parity-*N* Problems

	Parity-8	Parity-16	Parity-32	Parity-64
# inputs	8	16	32	64
# patterns	256	65536	4.294e+9	1.845e+19
MLP (one hidden layer)	9	17	33	65
BMLP (one hidden layer)	5	9	17	33
FCC	4	5	6	7

6.4 Recurrent Neural Networks

In contrast to feedforward neural networks, recurrent networks neuron outputs could be connected with their inputs. Thus, signals in the network can continuously be circulated. Until now, only a limited number of recurrent neural networks were described.

6.4.1 Hopfield Network

The single layer recurrent network was analyzed by Hopfield [H82]. This network shown in Figure 6.17 has unipolar hard threshold neurons with outputs equal to 0 or 1. Weights are given by a symmetrical square matrix **W** with zero elements ($w_{ij} = 0$ *for* $i=j$) on the main diagonal. The stability of the system is usually analyzed by means of the *energy function*

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}v_i v_j \tag{6.15}$$

It was proved that during signal circulation the energy E of the network decreases and system converges to the stable points. This is especially true when values of system outputs are updated in the asynchronous mode. This means that at the given cycle, only one random output can be changed to the required value. Hopfield also proved that those stable points to which the system converges can be programmed by adjusting the weights using a modified Hebbian [H49] rule

$$\Delta w_{ij} = \Delta w_{ji} = (2v_i - 1)(2v_j - 1) \tag{6.16}$$

Such memory has limited storage capacity. Based on experiments, Hopfield estimated that the maximum number of stored patterns is 0.15N, where N is the number of neurons.

6.4.2 Autoassociative Memory

Hopfield [H82] extended the concept of his network to autoassociative memories. In the same network structure as shown in Figure 6.19, the bipolar neurons were used with outputs equal to –1 of +1. In this network pattern, $\mathbf{s}_m$ are stored into the weight matrix **W** using autocorrelation algorithm

$$\mathbf{W} = \sum_{m=1}^{M} \mathbf{s}_m \mathbf{s}_m^T - M\mathbf{I} \tag{6.17}$$

where

M is the number of stored pattern
I is the unity matrix

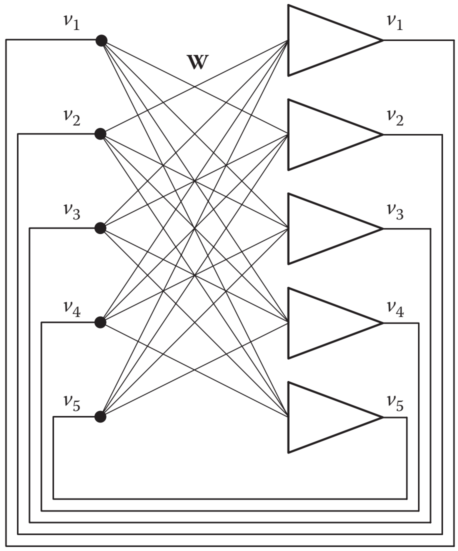


FIGURE 6.19 Autoassociative memory.

Note that **W** is the square symmetrical matrix with elements on the main diagonal equal to zero (w_{ji} *for* $i=j$). Using a modified formula, new patterns can be added or subtracted from memory. When such memory is exposed to a binary bipolar pattern by enforcing the initial network states, then after signal circulation the network will converge to the closest (most similar) stored pattern or to its complement.

This stable point will be at the closest minimum of the energy function

$$E(v) = -\frac{1}{2}\mathbf{v}^T\mathbf{W}\mathbf{v} \tag{6.18}$$

Like the Hopfield network, the autoassociative memory has limited storage capacity, which is estimated to be about M_{max}=0.15N. When the number of stored patterns is large and close to the memory capacity,

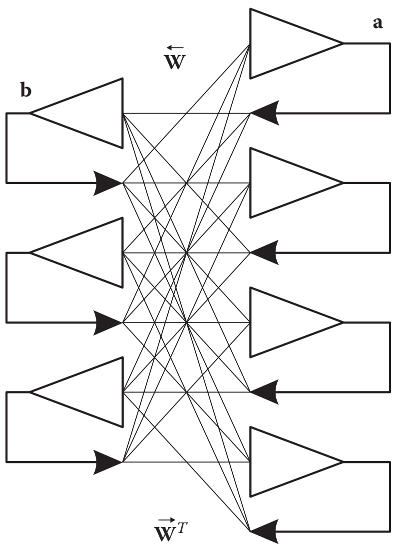


FIGURE 6.20 Bidirectional autoassociative memory.

the network has a tendency to converge to spurious states which were not stored. These spurious states are additional minima of the energy function.

6.4.3 BAM—Bidirectional Autoassociative Memories

The concept of the autoassociative memory was extended to bidirectional associative memories BAM by Kosko [87K]. This memory shown in Figure 6.20 is able to associate pairs of the patterns **a** and **b**.

This is the two layer network with the output of the second layer connected directly to the input of the first layer. The weight matrix of the second layer is $\mathbf{W}^T$ and it is $\mathbf{W}$ for the first layer. The rectangular weight matrix $\mathbf{W}$ is obtained as the sum of the cross correlation matrixes

$$\mathbf{W} = \sum_{m=1}^{M} \mathbf{a}_m \mathbf{b}_m \tag{6.19}$$

where

M is the number of stored pairs

$\mathbf{a}_m$ and $\mathbf{b}_m$ are the stored vector pairs

The BAM concept can be extended for association of three or more vectors.

References

[DB04] H. Demuth and M. Beale, *Neural Network Toolbox, User's Guide*, Version 4. The MathWorks, Inc., Natick, MA, revised for version 4.0.4 edition, October 2004. http://www.mathworks.com

[FL90] S. E. Fahlman and C. Lebiere, The cascade-correlation learning architecture. In D. S. Touretzky (ed.) *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, CA, 1990, pp. 524–532.

[H49] D. O. Hebb, *The Organization of Behavior, a Neuropsychological Theory*. Wiley, New York, 1949.

[H82] J. J. Hopfield, Neural networks and physical systems with emergent collective computation abilities. *Proceedings of the National Academy of Science* 79:2554–2558, 1982.

[H99] S. Haykin, *Neural Networks—A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, NJ, 1999.

[HN87] R. Hecht-Nielsen, Counterpropagation networks. *Applied Optics* 26(23):4979–4984, 1987.

[HW09] H. Yu and B. M. Wilamowski, C++ implementation of neural networks trainer. *13th International Conference on Intelligent Engineering Systems* (*INES-09*), Barbados, April 16–18, 2009.

[K87] B. Kosko, Adaptive bidirectional associative memories. *App. Opt*. 26:4947–4959, 1987.

[K88] T. Kohonen, The neural phonetic typerater. *IEEE Computer* 27(3):11–22, 1988.

[OW99] Y. Ota and B. M. Wilamowski, Analog implementation of pulse-coupled neural networks. *IEEE Transaction on Neural Networks* 10(3):539–544, May 1999.

[P89] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley Publishing Co., Reading, MA, 1989.

[W02] B. M. Wilamowski, Neural networks and fuzzy systems, Chap 32. In Robert R. Bishop (ed.) *Mechatronics Handbook*. CRC Press, Boca Raton, FL, 2002, pp. 33-1–32-26.

[W03] B. M. Wilamowski, Neural network architectures and learning (tutorial). *International Conference on Industrial Technology* (*ICIT'03*), Maribor, Slovenia, December 10–12, 2003.

[W07] B. M. Wilamowski, Neural networks and fuzzy systems for nonlinear applications. *11th International Conference on Intelligent Engineering Systems* (*INES 2007*), Budapest, Hungary, June 29–July 1, 2007, pp. 13–19.

[W09] B. M. Wilamowski, Neural network architectures and learning algorithms. *IEEE Industrial Electronics Magazine* 3(4):56–63.

[W89] P. D. Wasserman, *Neural Computing Theory and Practice*. Van Nostrand Reinhold, New York, 1989.

[W96] B. M. Wilamowski, Neural networks and fuzzy systems, Chaps. 124.1–124.8. In *The Electronic Handbook*. CRC Press, 1996, pp. 1893–1914.

[WB01] B. M. Wilamowski and J. Binfet, Microprocessor implementation of fuzzy systems and neural networks. *International Joint Conference on Neural Networks* (*IJCNN'01*), Washington, DC, July 15–19, 2001, pp. 234–239.

[WCKD07] B. M. Wilamowski, N. J. Cotton, O. Kaynak, and G. Dundar, Method of computing gradient vector and Jacobian matrix in arbitrarily connected neural networks. *IEEE International Symposium on Industrial Electronics* (*ISIE 2007*), Vigo, Spain, June 4–7, 2007, pp. 3298–3303.

[WCKD08] B. M. Wilamowski, N. J. Cotton, O. Kaynak, and G. Dundar, Computing gradient vector and Jacobian matrix in arbitrarily connected neural networks. *IEEE Transactions on Industrial Electronics* 55(10):3784–3790, October 2008.

[WH10] B. M. Wilamowski and H. Yu, Improved computation for Levenberg Marquardt training. *IEEE Transactions on Neural Networks* 21:930–937, 2010.

[WHM03] B. M. Wilamowski, D. Hunter, and A. Malinowski, Solving parity-*N* problems with feedforward neural network. *Proceedings of the International Joint Conference on Neural Networks* (*IJCNN'03*), Portland, OR, July 20–23, 2003, pp. 2546–2551.

[WJ96] B. M. Wilamowski and R. C. Jaeger, Implementation of RBF type networks by MLP networks. *IEEE International Conference on Neural Networks*, Washington, DC, June 3–6, 1996, pp. 1670–1675.

[WJK99] B. M. Wilamowski, R. C. Jaeger, and M. O. Kaynak, Neuro-fuzzy architecture for CMOS implementation. *IEEE Transaction on Industrial Electronics* 46(6):1132–1136, December 1999.

[WJPM96] B. M. Wilamowski, R. C. Jaeger, M. L. Padgett, and L. J. Myers, CMOS implementation of a pulse-coupled neuron cell. *IEEE International Conference on Neural Networks*, Washington, DC, June 3–6, 1996, pp. 986–990.

[WPJ96] B. M. Wilamowski, M. L. Padgett, and R. C. Jaeger, Pulse-coupled neurons for image filtering. *World Congress of Neural Networks*, San Diego, CA, September 15–20, 1996, pp. 851–854.

[Z92] J. Zurada, *Introduction to Artificial Neural Systems*. West Publishing Company, St. Paul, MN, 1992.

[Z95] J. M. Zurada, *Artificial Neural Systems*. PWS Publishing Company, St. Paul, MN, 1995.