5

Understanding of Neural Networks

Bogdan M. Wilamowski
Auburn University

5.1 Introduction **5**-1
5.2 The Neuron **5**-1
5.3 Should We Use Neurons with Bipolar or Unipolar Activation Functions? **5**-5
5.4 Feedforward Neural Networks **5**-5
References **5**-10

5.1 Introduction

The fascination of artificial neural networks started in the middle of the previous century. First artificial neurons were proposed by McCulloch and Pitts [MP43] and they showed the power of the threshold logic. Later Hebb [H49] introduced his *learning rules*. A decade later, Rosenblatt [R58] introduced the perceptron concept. In the early 1960s, Widrow and Holf [WH60] developed intelligent systems such as ADALINE and MADALINE. Nilsson [N65] in his book, *Learning Machines*, summarized many developments of that time. The publication of the Mynsky and Paper [MP69] book, with some discouraging results, stopped for sometime the fascination with artificial neural networks, and achievements in the mathematical foundation of the backpropagation algorithm by Werbos [W74] went unnoticed. The current rapid growth in the area of neural networks started with the work of Hopfield's [H82] recurrent network, Kohonen's [K90] unsupervised training algorithms, and a description of the backpropagation algorithm by Rumelhart et al. [RHW86]. Neural networks are now used to solve many engineering, medical, and business problems [WK00,WB01,B07,CCBC07,KTP07,KT07,MFP07,FP08,JM08,W09]. Descriptions of neural network technology can be found in many textbooks [W89,Z92,H99,W96]. AQ1

5.2 The Neuron

A biological neuron is a complicated structure, which receives trains of pulses on hundreds of *excitatory* and *inhibitory* inputs. Those incoming pulses are summed with different weights (averaged) during the time period [WPJ96]. If the summed value is higher than a threshold, then the neuron itself is generating a pulse, which is sent to neighboring neurons. Because incoming pulses are summed with time, the neuron generates a pulse train with a higher frequency for higher positive excitation. In other words, if the value of the summed weighted inputs is higher, the neuron generates pulses more frequently. At the same time, each neuron is characterized by the nonexcitability for a certain time after the firing pulse. This so-called *refractory period* can be more accurately described as a phenomenon, where after excitation, the threshold value increases to a very high value and then decreases gradually with a certain time constant. The refractory period sets soft upper limits on the frequency of the output pulse train. In the biological neuron, information is sent in the form of frequency-modulated pulse trains.

K10149_C005.indd 1 8/31/2010 4:32:01 AM

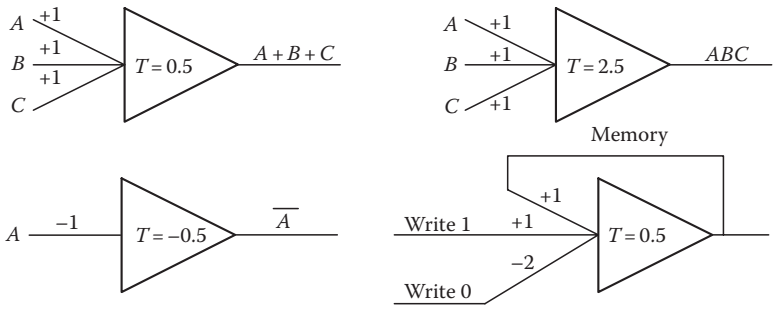


FIGURE 5.1 Examples of logical operations using McCulloch–Pitts neurons.

The description of neuron action leads to a very complex neuron model, which is not practical. McCulloch and Pitts [MP43] show that even with a very simple neuron model, it is possible to build logic and memory circuits. Examples of McCulloch-Pitts' neurons realizing OR, AND, NOT, and MEMORY operations are shown in Figure 5.1.

Furthermore, these simple neurons with thresholds are usually more powerful than typical logic gates used in computers (Figure 5.1). Note that the structure of OR and AND gates can be identical. With the same structure, other logic functions can be realized, as shown in Figure 5.2.

The McCulloch-Pitts neuron model (Figure 5.3a) assumes that incoming and outgoing signals may have only binary values 0 and 1. If incoming signals summed through positive or negative weights have a value equal or larger than threshold, then the neuron output is set to 1. Otherwise, it is set to 0.

$$out = \begin{cases} 1 & \text{if } net \geq T \\ 0 & \text{if } net < T \end{cases} \tag{5.1}$$

where

T is the threshold

net value is the weighted sum of all incoming signals (Figure 5.3)

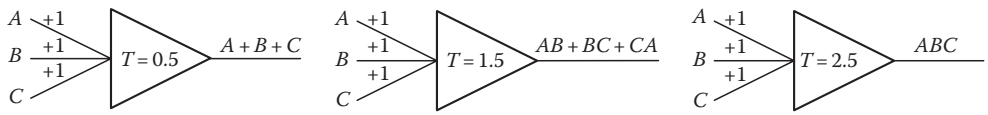


FIGURE 5.2 The same neuron structure and the same weights, but a threshold change results in different logical functions.

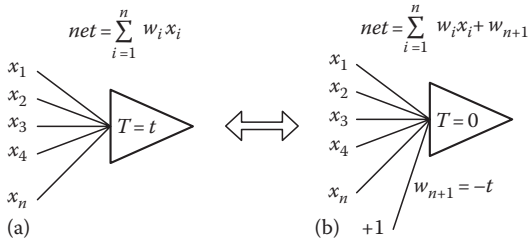


FIGURE 5.3 Threshold implementation with an additional weight and constant input with +1 value: (a) neuron with threshold T and (b) modified neuron with threshold $T = 0$ and additional weight $w_{n+1} = -t$.

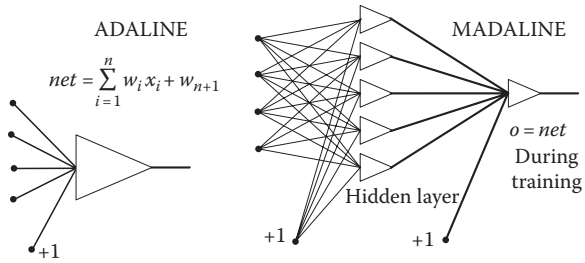


FIGURE 5.4 ADALINE and MADALINE perceptron architectures.

The perceptron model has a similar structure (Figure 5.3b). Its input signals, the weights, and the thresholds could have any positive or negative values. Usually, instead of using variable threshold, one additional constant input with a negative or positive weight can be added to each neuron, as Figure 5.3 shows. Single-layer perceptrons are successfully used to solve many pattern classification problems. Most known perceptron architectures are ADALINE and MADALINE [WH60] shown in Figure 5.4.

Perceptrons using hard threshold activation functions for unipolar neurons are given by

$$o = f_{uni}(net) = \frac{\operatorname{sgn}(net)+1}{2} = \begin{cases} 1 & \text{if } net \geq 0 \\ 0 & \text{if } net < 0 \end{cases} \tag{5.2}$$

and for bipolar neurons

$$o = f_{bip}(net) = \operatorname{sgn}(net) = \begin{cases} 1 & \text{if } net \geq 0 \\ -1 & \text{if } net < 0 \end{cases} \tag{5.3}$$

For these types of neurons, most of the known training algorithms are able to adjust weights only in single-layer networks. Multilayer neural networks (as shown in Figure 5.8) usually use soft activation functions, either unipolar

$$o = f_{uni}(net) = \frac{1}{1+\exp(-\lambda net)} \tag{5.4}$$

or bipolar

$$o = f_{bip}(net) = \tanh(0.5\lambda net) = \frac{2}{1+\exp(-\lambda net)} - 1 \tag{5.5}$$

These soft activation functions allow for the gradient-based training of multilayer networks. Soft activation functions make neural network transparent for training [WT93]. In other words, changes in weight values always produce changes on the network outputs. This would not be possible when hard activation

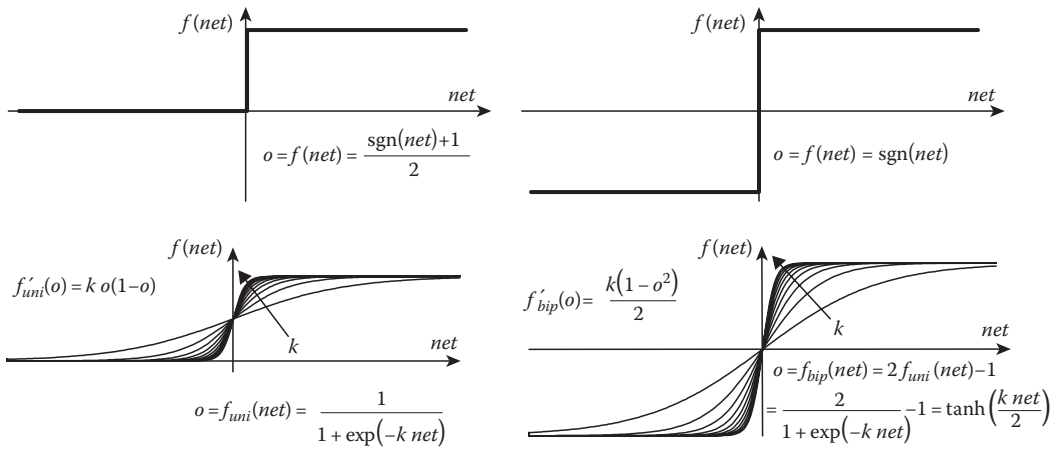


FIGURE 5.5 Typical activation functions: hard in upper row and soft in the lower row.

functions are used. Typical activation functions are shown in Figure 5.5. Note, that even neuron models with continuous activation functions are far from an actual biological neuron, which operates with frequency-modulated pulse trains [WJPM96].

A single neuron is capable of separating input patterns into two categories, and this separation is linear. For example, for the patterns shown in Figure 5.6, the separation line is crossing x_1 and x_2 axes at points x_{10} and x_{20}. This separation can be achieved with a neuron having the following weights: $w_1 = 1/x_{10}$; $w_2 = 1/x_{20}$ and $w_3 = -1$. In general, for n dimensions, the weights are

$$w_i = \frac{1}{x_{i0}} \qquad \text{for } i = 1,\ldots,n; \quad w_{n+1} = -1$$

One neuron can divide only linearly separated patterns. To select just one region in n-dimensional input space, more than $n + 1$ neurons should be used.

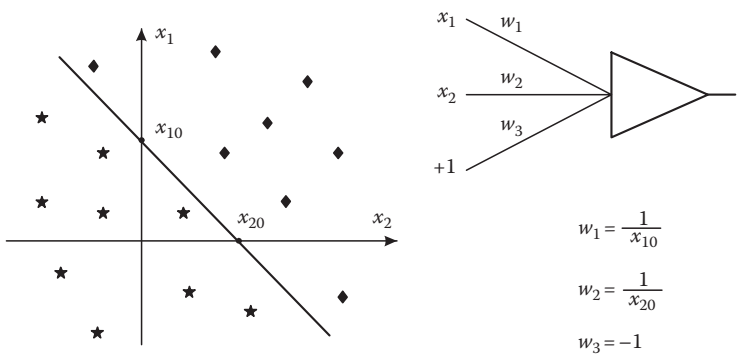


FIGURE 5.6 Illustration of the property of linear separation of patterns in the two-dimensional space by a single neuron.

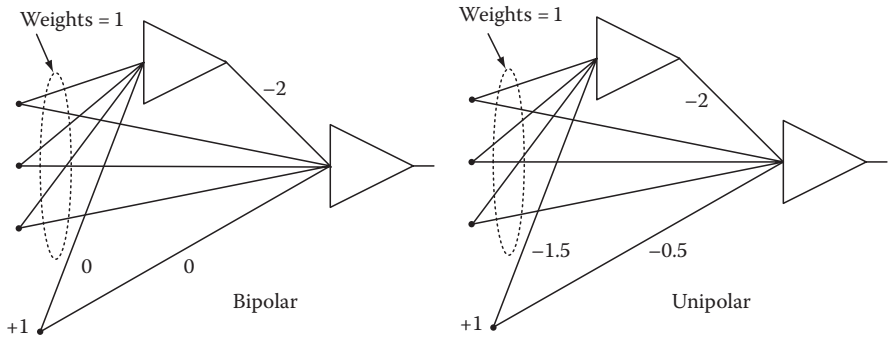


FIGURE 5.7 Neural networks for parity-3 problem.

5.3 Should We Use Neurons with Bipolar or Unipolar Activation Functions?

Neural network users often face a dilemma if they have to use unipolar or bipolar neurons (see Figure 5.5). The short answer is that it does not matter. Both types of networks work the same way and it is very easy to transform bipolar neural network into unipolar neural network and vice versa. Moreover, there is no need to change most of weights but only the biasing weight has to be changed. In order to change from bipolar networks to unipolar networks, only biasing weights must be modified using the formula

$$w_{bias}^{uni} = 0.5\left(w_{bias}^{bip} - \sum_{i=1}^{N} w_i^{bip} \right) \tag{5.6}$$

While, in order to change from unipolar networks to bipolar networks

$$w_{bias}^{bip} = 2w_{bias}^{uni} + \sum_{i=1}^{N} w_i^{uni} \tag{5.7}$$

Figure 5.7 shows the neural network for parity-3 problem, which can be transformed both ways: from bipolar to unipolar and from unipolar to bipolar. Notice that only biasing weights are different. Obviously input signals in bipolar network should be in the range from −1 to +1, while for unipolar network they should be in the range from 0 to +1.

5.4 Feedforward Neural Networks

Feedforward neural networks allow only unidirectional signal flow. Furthermore, most feedforward neural networks are organized in layers and this architecture is often known as MLP (multilayer perceptron). An example of the three-layer feedforward neural network is shown in Figure 5.8. This network consists of four input nodes, two *hidden layers*, and an output layer.

If the number of neurons in the input (hidden) layer is not limited, then all classification problems can be solved using a multilayer network. An example of such neural network, separating patterns from the rectangular area on Figure 5.9 is shown in Figure 5.10

When the hard threshold activation function is replaced by soft activation function (with a gain of 10), then each neuron in the hidden layer will perform a different task as it is shown in Figure 5.11 and the

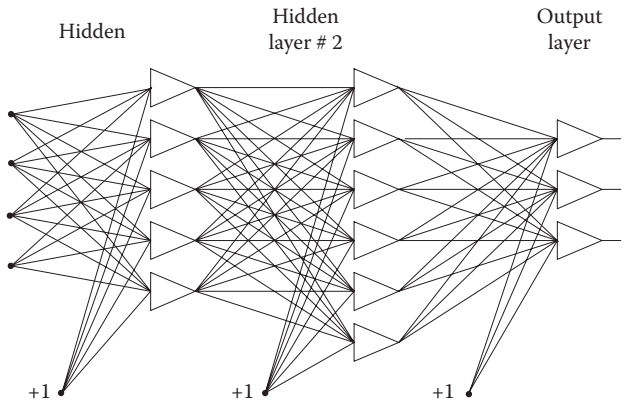


FIGURE 5.8 An example of the three-layer feedforward neural network, which is sometimes known also as MLP.

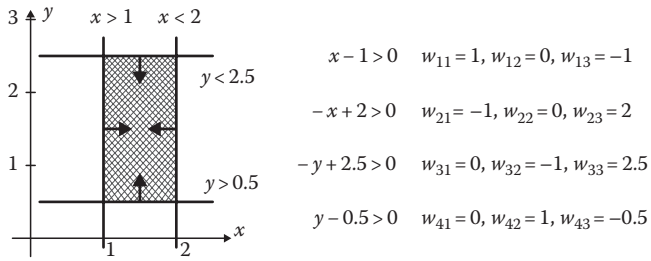


FIGURE 5.9 Rectangular area can be separated by four neurons.

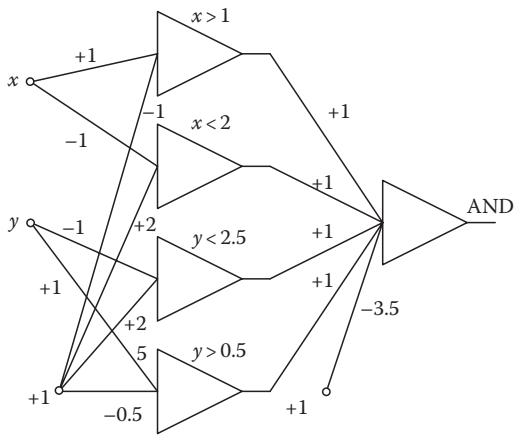


FIGURE 5.10 Neural network architecture that can separate patterns in the rectangular area of Figure 5.7.

response of the output neuron is shown in Figure 5.12. One can notice that the shape of the output surface depends on the gains of activation functions. For example, if this gain is set to be 30, then activation function looks almost as hard activation function and the neural network work as a classifier (Figure 5.13a). If the neural network gain is set to a smaller value, for example, equal 5, then the neural network performs a nonlinear mapping, as shown in Figure 5.13b. Even though this is a relatively simple example, it is essential for understanding neural networks.

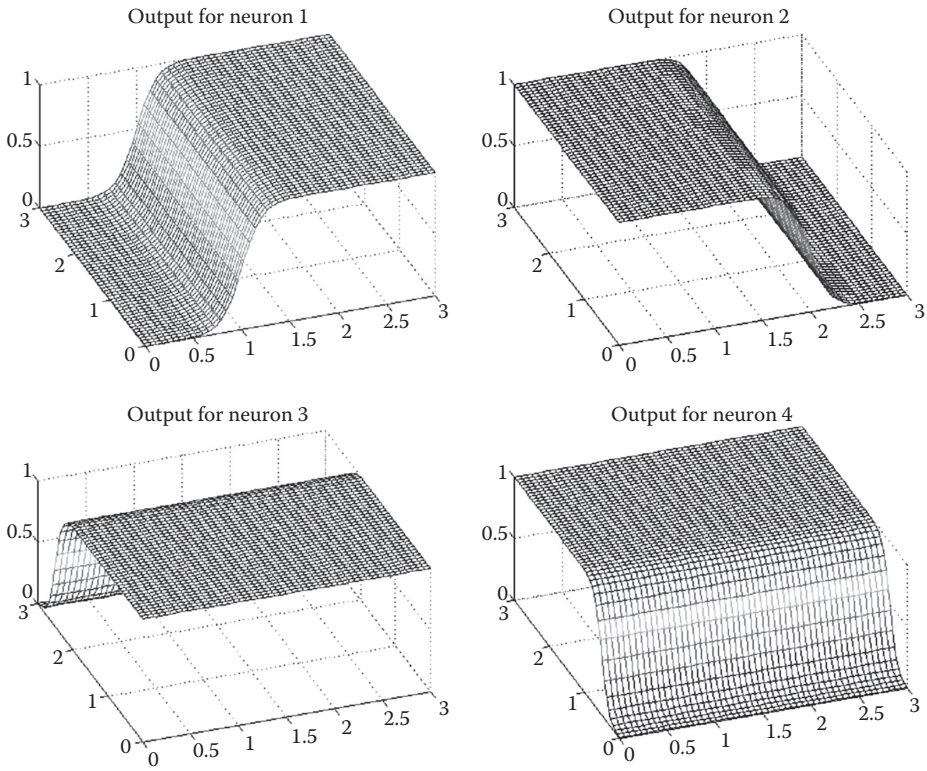


FIGURE 5.11 Responses of four hidden neurons of the network from Figure 5.10.

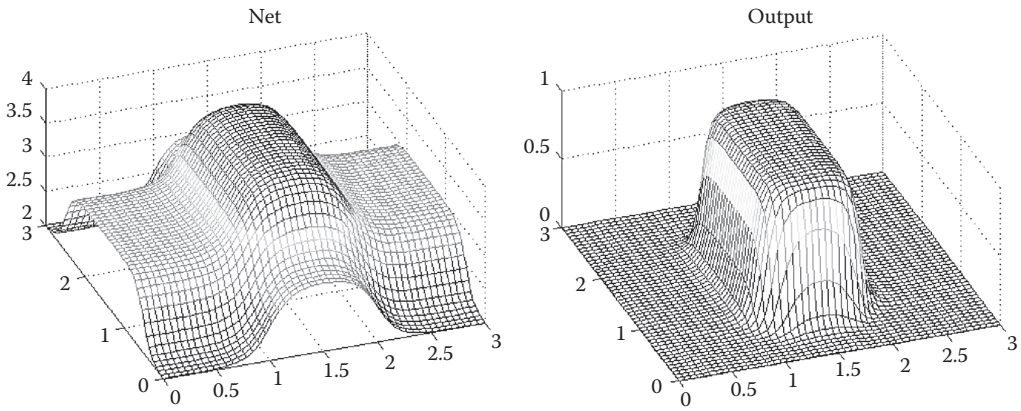


FIGURE 5.12 The net and output values of the output neuron of the network from Figure 5.10.

Let us now use the same neural network architecture as shown in Figure 5.10, but let us change weights for hidden neurons so their neuron lines are located as it is shown in Figure 5.14. This network can separate patterns in pentagonal shape as shown in Figure 5.15a or perform a complex nonlinear mapping as shown in Figure 5.15b depending on the neuron gains. In this simple example of network from Figure 5.10, it is very educational because it lets neural network user understand how neural network operates and may help to select a proper neural network architecture for problems of different complexities. Commonly used trial-and-error methods may not be successful unless the user may have some understanding of neural network operation.

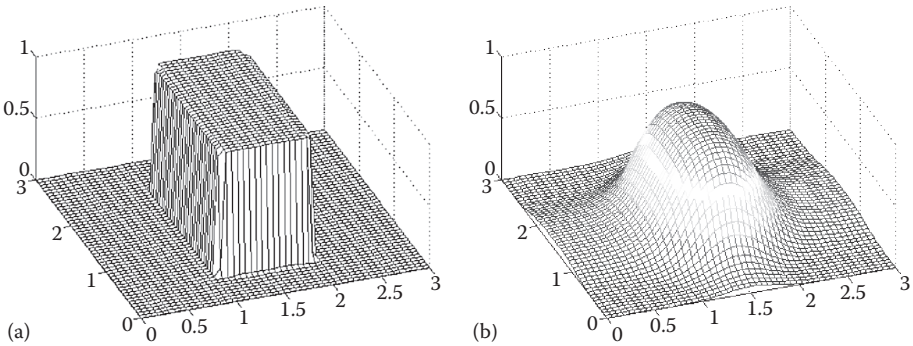


FIGURE 5.13 Response on the neural network of Figure 5.10 with different values of neurons gain: (a) gain = 30 and network works as classifier and (b) gain = 5 and network perform nonlinear mapping.

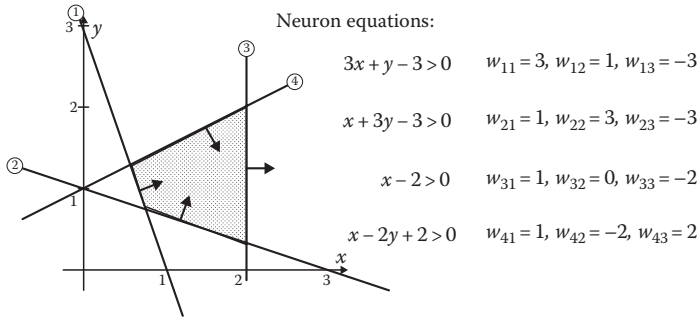


FIGURE 5.14 Two-dimensional input space with four separation lines representing four neurons.

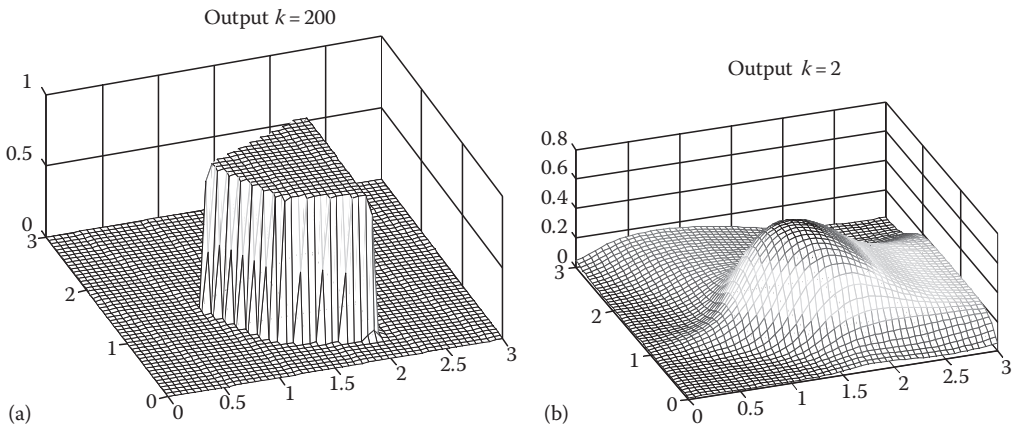
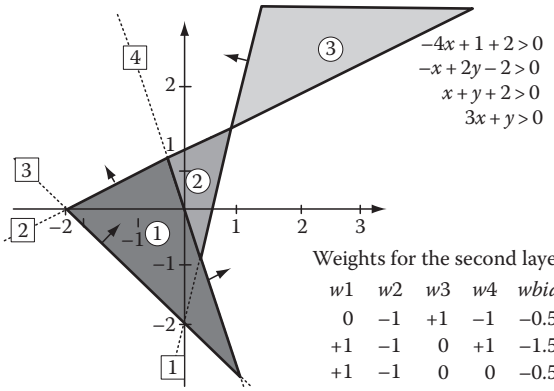


FIGURE 5.15 Response on the neural network of Figure 5.9 with weights define in Figure 5.13 for different values of neurons gain: (a) gain = 200 and network works as classifier and (b) gain = 2 and network perform nonlinear mapping.



Weights for the first layer:

wx	*wy*	*wbias*	
−4	1	2	*layer*1_ *neuron*1
−1	2	−2	*layer*1_ *neuron*2
1	1	2	*layer*1_ *neuron*3
3	1	0	*layer*1_ *neuron*3

Weights for the second layer:

*w*1	*w*2	*w*3	*w*4	*wbias*	
0	−1	+1	−1	−0.5	*layer*2_ *neuron*1
+1	−1	0	+1	−1.5	*layer*2_ *neuron*2
+1	−1	0	0	−0.5	*layer*2_ *neuron*3

FIGURE 5.16 Problem with the separation of three clusters.

The linear separation property of neurons makes some problems especially difficult for neural networks, such as exclusive OR, parity computation for several bits, or to separate patterns on two neighboring spirals. Also, the most commonly used feedforward neural network may have difficulties to separate clusters in multidimensional space. For example, in order to separate cluster in two-dimensional space, we have used four neurons (rectangle), but it is also possible to separate cluster with three neurons (triangle). In three dimensions we may need at least four planes (neurons) to separate space with tetrahedron. In *n*-dimensional space, in order to separate a cluster of

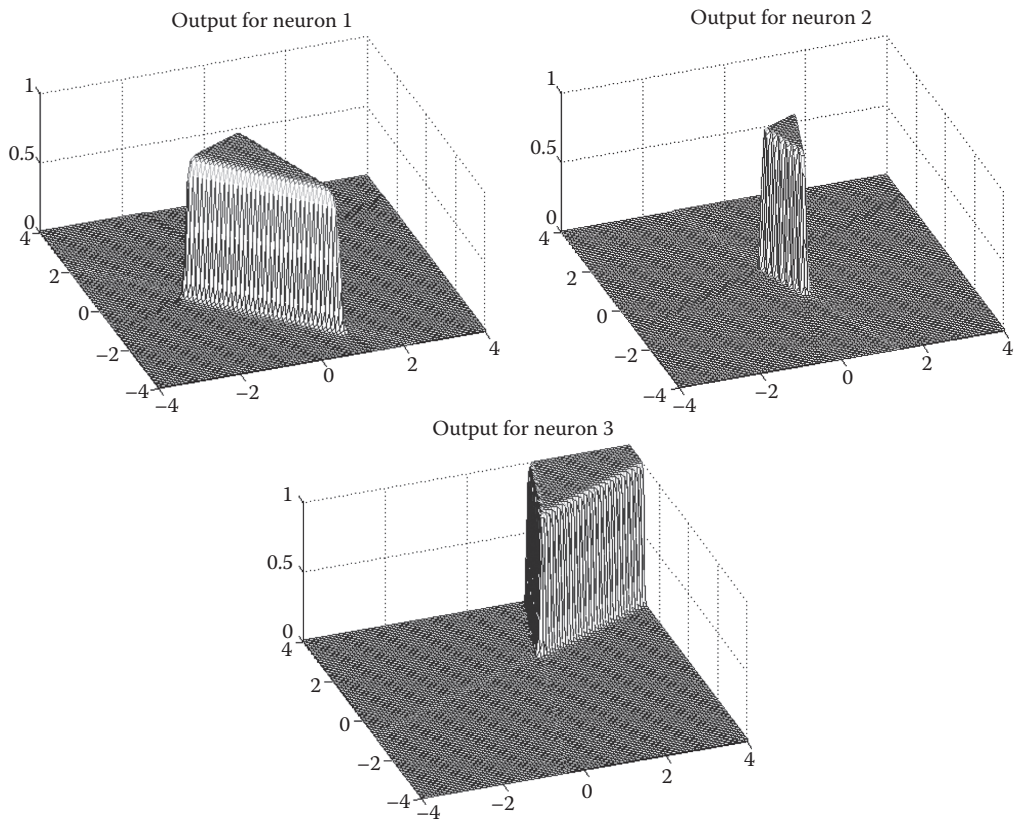


FIGURE 5.17 Neural Network performing cluster separation and resulted output surfaces for all three clusters.

patterns, there are at least $n + 1$ neurons required. However, if neural network with several hidden layers are used, then the number of neurons needed may not be that excessive. Also, a neuron in the first hidden layer may be used for separation of multiple clusters. Let us analyze another example where we would like to design neural network with multiple outputs to separate three clusters and each network output must produce +1 only for a given cluster. Figure 5.16 shows three clusters to be separated, corresponding equations for four neurons and weights for resulted neural network, as shown in Figure 5.17.

The example with three clusters shows that often there is no need to have several neurons in the hidden layer dedicated for a specific cluster. These hidden neurons may perform multiple functions and they can contribute to several clusters instead of just one. It is, of course, possible to develop separate neural networks for every cluster, but it is much more efficient to have one neural network with multiple outputs as shown in Figures 5.16 and 5.17. This is one advantage of neural networks over fuzzy systems, AQ2 which can be developed only for one output at a time [WJK99]. Another advantage of neural network is that the number of inputs can be very large so they can process signals in multidimensional space, while fuzzy systems can handle usually two or three inputs only [WB99].

The most commonly used neural networks have the MLP architecture, as shown in Figure 5.8. For such a layer-by-layer network, it is relatively easy to develop the learning software, but these networks are significantly less powerful than networks where connections across layers are allowed. Unfortunately, only very limited number of software were developed to train other than MLP networks [WJ96,W02]. As a result, most researchers use MLP architectures, which are far from optimal. Much better results can be obtained with BMLP (bridged MLP) architecture or with FCC (fully connected cascade) architecture [WHM03]. Also, most researchers are using simple EBP (error backpropagation) learning algorithm, which is not only much slower than more advanced algorithms such as LM (Levenberg–Marquardt) [HM94] or NBN (Neuron by Neuron) [WCKD08,HW09,WH10], but also EBP algorithm often is not able to train close-to-optimal neural networks [W09].

References

[B07] B.K. Bose, Neural network applications in power electronics and motor drives—An introduction and perspective. *IEEE Trans. Ind. Electron*. 54(1):14–33, February 2007.

[CCBC07] G. Colin, Y. Chamaillard, G. Bloch, and G. Corde, Neural control of fast nonlinear systems—Application to a turbocharged SI engine with VCT. *IEEE Trans. Neural Netw*. 18(4):1101–1114, April 2007.

[FP08] J.A. Farrell and M.M. Polycarpou, Adaptive approximation based control: Unifying neural, fuzzy and traditional adaptive approximation approaches. *IEEE Trans. Neural Netw*. 19(4):731–732, April 2008.

[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. *Proc. Natl. Acad. Sci*. 79:2554–2558, 1982.

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

[HM94] M.T. Hagan and M. Menhaj, Training feedforward networks with the Marquardt algorithm. *IEEE Trans. Neural Netw*. 5(6):989–993, 1994.

[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.

[JM08] M. Jafarzadegan and H. Mirzaei, A new ensemble based classifier using feature transformation for hand recognition. *2008 Conference on Human System Interactions*, Krakow, Poland, May 2008, pp. 749–754.

[K90] T. Kohonen, The self-organized map. *Proc. IEEE* 78(9):1464–1480, 1990.

[KT07] S. Khomfoi and L.M. Tolbert, Fault diagnostic system for a multilevel inverter using a neural network. *IEEE Trans. Power Electron*. 22(3):1062–1069, May 2007.

[KTP07] M. Kyperountas, A. Tefas, and I. Pitas, Weighted piecewise LDA for solving the small sample size problem in face verification. *IEEE Trans. Neural Netw*. 18(2):506–519, February 2007.

[MFP07] J.F. Martins, V. Ferno Pires, and A.J. Pires, Unsupervised neural-network-based algorithm for an on-line diagnosis of three-phase induction motor stator fault. *IEEE Trans. Ind. Electron*. 54(1):259–264, February 2007.

[MP43] W.S. McCulloch and W.H. Pitts, A logical calculus of the ideas imminent in nervous activity. *Bull. Math. Biophy*. 5:115–133, 1943.

[MP69] M. Minsky and S. Papert, *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[MW01] M. McKenna and B.M. Wilamowski, Implementing a fuzzy system on a field programmable gate array. *International Joint Conference on Neural Networks* (*IJCNN'01*), Washington, DC, July 15–19, 2001, pp. 189–194. AQ3

[N65] N.J. Nilsson, *Learning Machines: Foundations of Trainable Pattern Classifiers*. McGraw Hill Book Co., New York, 1965.

[R58] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev*. 65:386–408, 1958.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Wiliams, Learning representations by back-propagating errors. *Nature* 323:533–536, 1986.

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

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

[W74] P. Werbos, Beyond regression: New tools for prediction and analysis in behavioral sciences. PhD dissertation, Harvard University, Cambridge, MA, 1974.

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

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

[WB99] B.M. Wilamowski and J. Binfet, Do fuzzy controllers have advantages over neural controllers in microprocessor implementation. *Proceedings of the 2nd International Conference on Recent Advances in Mechatronics - ICRAM'99*, Istanbul, Turkey, May 24–26, 1999, pp. 342–347.

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

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

[WH60] B. Widrow and M.E. Hoff, Adaptive switching circuits. *1960 IRE Western Electric Show and Convention Record, Part 4*, New York (August 23), pp. 96–104, 1960.

[WHM03] B. Wilamowski, D. Hunter, and A. Malinowski, Solving parity-N problems with feedforward neural network. *Proceedings of the IJCNN' 03 International Joint Conference on Neural Networks*, 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.

[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.

[WK00] B.M. Wilamowski and O. Kaynak, Oil well diagnosis by sensing terminal characteristics of the induction motor. *IEEE Trans. Ind. Electron*. 47(5):1100–1107, October 2000.

[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.

[WT93] B.M. Wilamowski and L. Torvik, Modification of gradient computation in the back-propagation algorithm. Presented at *ANNIE'93—Artificial Neural Networks in Engineering*, St. Louis, MO, November 14–17, 1993.

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

K10149_C005.indd 12 8/31/2010 4:32:37 AM