NEURAL NETS BASED ON BIOLOGY

The idea of building modular networks comes from the analogy with biological systems, in which a brain (as a common example) consists of a series of interconnected substructures, like auditory, vestibular, and visual systems, which in turn are further structured on more functionally independent groups of neurons. Each level of signal processing performs its unique and independent purpose, so that the complexity of the output of each subsystem depends on the hierarchical level of that subsystem within the whole system. For instance, in the striate cortex (area 17 of Broadman's areas of the brain), simple cells provide increased activity when a bar or slit of light stimulates a precise area of the visual field at a precise orientation. The cell output is further processed by complex neurons, which respond best to straight lines moving through the receptive field in a particular direction with a specific orientation. Therefore, the dot-like information from ganglion and lateral geniculate cells is transformed in the occipital lobe into information about edges and their position, length, orientation, and movement. Although this represents a high degree of abstraction, the visual association areas of the occipital lobe are only an early stage in the integration of visual information.

Modular neural networks are used in a broad variety of applications. One example is character recognition. In Fig. 1(a) three levels of modules are represented. Each module has

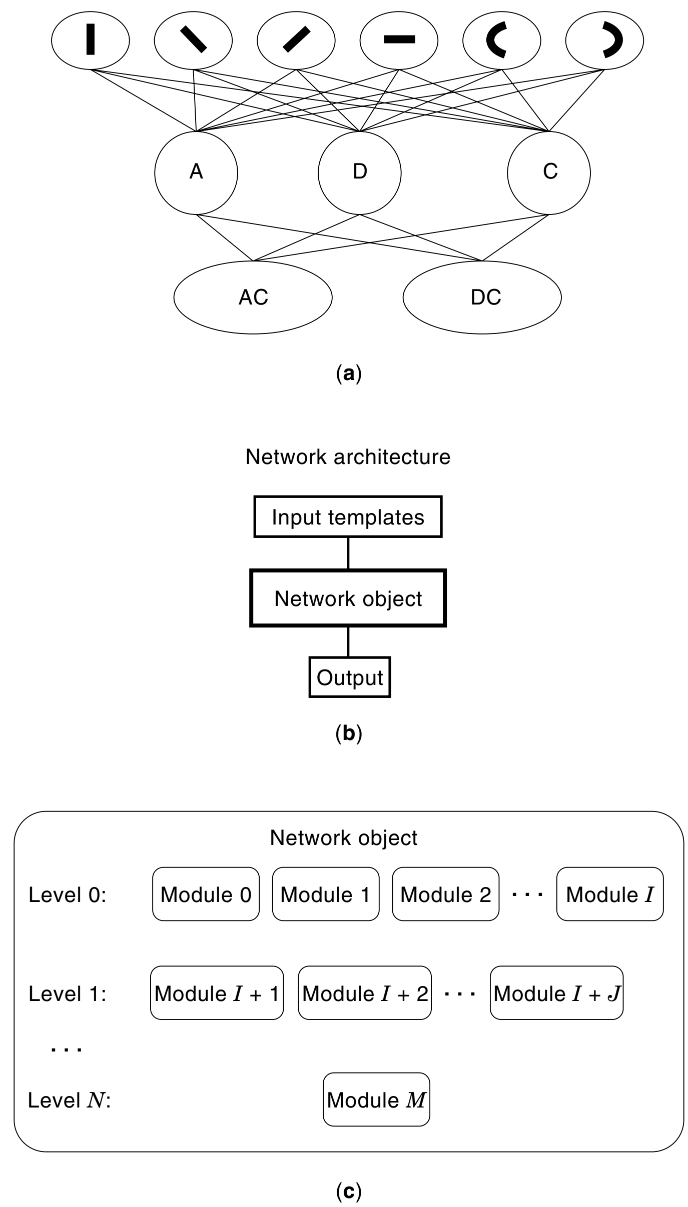


Figure 1. An example of a modular neural network for character recognition. (a) The modules at the first level recognize specific features, and the modules at the second level recognize letters, which are made of the features from the first level. The modules at the third level recognize words made of letters from the second level. (b) General neural network architecture. (c) A network object consists of a container of module objects. The intermodular connectivity is a property of each module. The object also stores some general parameters common to all network's counterparts (i.e., parameters of noise, min and max boundaries of weight, max value of weight change, etc.)

its unique function, providing some output to the modules in the next level. In this case the modules in the top level recognize the specified features of the letters A, D, and C, the modules at the second level recognize the letters themselves, and modules of the third level recognize groups of letters, and so on. The usage of modular neural networks is most beneficial when there are cases of missing pieces of data. Because each module takes its input from several others, a missing connection between modules would not significantly alter that module's output.

NETWORK ARCHITECTURE

At a level of high abstraction the network should look like an object in Fig. 1(b). It receives some input from templates stored in a file, propagates it through all modules, and provides some output in a meaningful format.

The "network object" consists of a series of levels of modules, similar to Fig. 1(a). It also provides some common data for each module, such as mean and sigma values for noise, max weight, and bound.

A more detailed representation of a network object is given in Fig. 1(c). This example contains N levels of modules Level 0 to Level N. Initially each module at a current level is connected to all modules in the next level. However, it is possible to remove certain intermodular connections to make the network more problem-oriented. Each module has an identifier specific to the level at which the module is located.

Each module has two containers. In the first container it stores all layers of neurons. In the second container it keeps track of all connections to other modules within the network. The information in both containers is specific to this particular module. A module is not aware of any type of processing going on in the rest of the network, though it knows which particular network it belongs to in order to provide correct references (stored in the second container) to the rest of the modules.

Each module also has two additional arrays, where it stores its responses to each template in the training set and where it saves its output after each iteration during training and after computing the outputs during the testing cycle.

A connection of a module is simply an integral reference equal to the destination module index in the network's container of all modules. A container of layers consists of the layer objects [Fig. 2(a)].

A layer object consists of a container of node objects. It is just a framework for efficient storage of nodes to ease the propagation of signal through this module during training.

A node object (neuron) is shown in Fig. 2(b). It consists of a container of connection objects and a place to store a current "charge" that is accumulated by the node. This charge is equivalent to the one seen by the axon hillock in a biological neuron. A node object knows how to add additional "charge" to itself, how to zero its "charge," and how to send this charge to all of its connection objects. This process is similar to the propagation of action potentials in a biological system.

The connection object is a general liaison between nodes. Its functionality is similar to a branch of a terminal dendrite of a neuron ending to a synapse. A connection object has a place to store current connection weights (and its old value used in training), and it maintains a delayed queue, which provides a mechanism for implementing a delay (in number of iterations) that occurs before a signal is detected at the synapse of this connection.

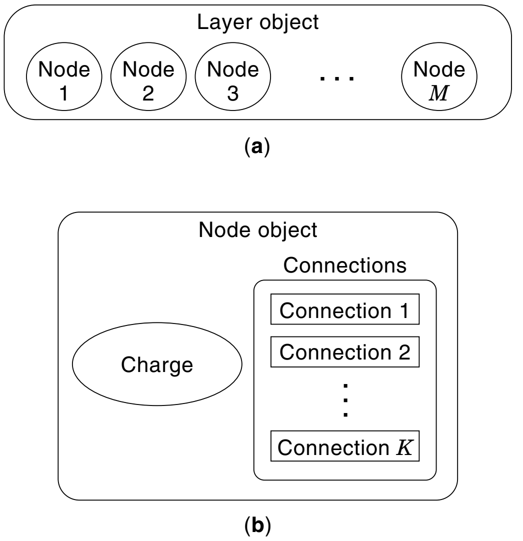


Figure 2. (a) A node object (neuron) has a "charge," analogous to the charge at the axon hillock in a biological neuron. The object has a container of all connections that provide references to other neurons which receive the "action potentials" (charges) generated by this node and propagated through each of these connections. (b) A layer object is a container of node objects (neurons).

The lateral inhibition is identified by the fact that the connection points to a node which is located on the same layer as the one that owns the connection:

1. The number of output nodes in all modules may equal the number of templates in the training set.
2. The output value x_i of any output node of each module is in range $0 \leq x_i < 1$.
3. While training, a desired output vector ($\boldsymbol{O}_i^{\text{desired}}$) of any module in the network can have only one dominant value per template (i.e., if the number of templates in the training set is four, then for the first template: $\boldsymbol{O}_1^{\text{desired}} = \{1,0,0,0\}$, for the second template: $\boldsymbol{O}_2^{\text{desired}} = \{0,1,0,0\}$, etc.).
4. The number of modules per network and layers/nodes/connections per module are limited mainly by the available memory of a computer.

EXECUTING THE NETWORK

The process of executing the network engine is described in three major steps:

1. Initializing
2. Training
3. Testing

Details are given in Appendix 1.

RESULTS

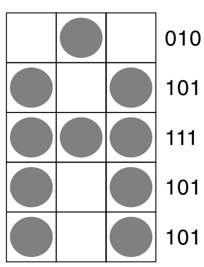
We performed three different tests on the modular network. The objective for the tests was to compare the difference in terms of the accuracy of recognition of templates containing missing features between a traditional network made of a single module and a network containing multiple modules. The single module network was used as a control in all tests.

In all cases the networks converged to 99% for the training set of templates. The testing set of templates was made of the training templates missing some features (a missing feature is defined as a feature whose value is 0).

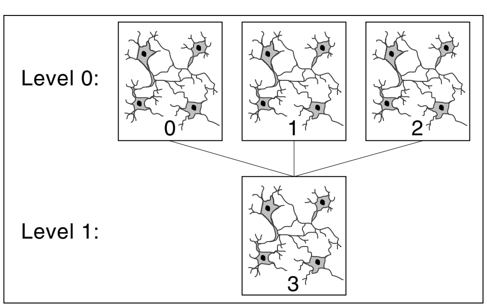
The first test was done on five templates in the training set. Each template consists of forty-five features representing three letters of the alphabet from A to F [Fig. 3(a)]. Five templates ABF, BCD, EDC, FEB, and DFA were used in the training set for both networks. The modular network was configured to have three modules in the input level and one module in the output level [Fig. 3(b)]. Each module in the input level received 15 features (representing one letter) per template. The control network consists of one module only, which received all 45 features per template as its input.

The testing cycle consists of fifteen templates: AB0, A0F, 0BF, BC0, B0D, 0CD, ED0, E0C, 0DC, FE0, F0B, 0EB, DF0, D0A, and 0FA. The control network recognized three templates as "similar" to the templates from the training set, whereas the modular network recognized thirteen.

In the second test we used six templates containing sixteen features each. The features were arranged in four groups of four-bit binary representations of digits from 1 to 15: 7 9 13 14, 9 13 14 15, 13 14 15 3, 14 15 3 9, 15 3 9 11, and 3 7 11 13. Each of these digits (four-bit representations) was inputted to

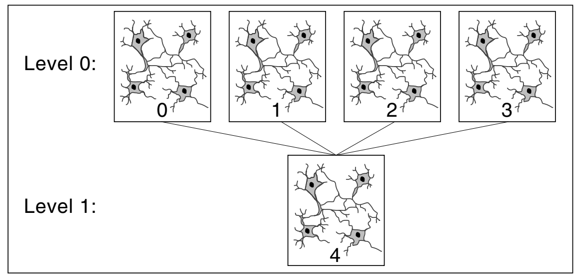


(a)

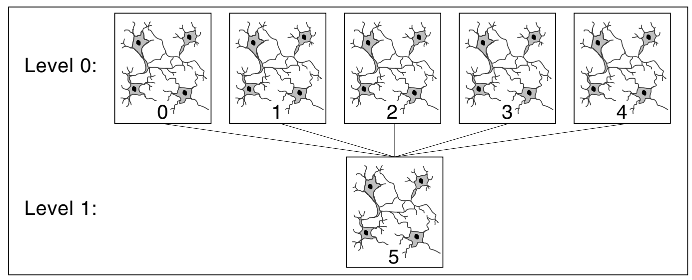


(b)

Figure 3. (a) A template for the first test contains a binary representation of alphabetic letters A through F. Each letter is encoded by 15 bits, as shown in the figure. (b) The configuration of the network in the first test.



(a)



(b)

Figure 4. (a) The configuration of the network in the second test. (b) The configuration of the network in the third test.

each module at the input level of the network [Fig. 4(a)]. The control network received all sixteen features per template as its input.

The testing set consists of 24 templates derived from each template in the training set by omitting one digit (making its four-bit representation equal to 0000).

The control network recognized 11 templates, whereas the modular network recognized 20 templates.

In the third test we used seven templates containing fifteen features each. The features were arranged in five groups of three-bit binary representations of digits from 1 to 7. The templates were 1 2 3 4 5, 2 3 4 5 6, 3 4 5 6 7, 4 5 6 7 1, 5 6 7 1 2, 6 7 1 2 3, and 7 1 2 3 4. The modular network consists of five modules in the input level (receiving one digit, i.e., three features per module) and one module in the output level [Fig. 4(b)]. The control network received all fifteen features per template.

The testing set contains 35 templates derived from the templates in the training set in the same way as in the second test.

The control network recognized 24 templates, whereas the modular network recognized 29 templates.

DISCUSSION

The networks showed convergence of 99% in all tests which we performed. The tests demonstrate a direct advantage of using modular networks when one deals with missing features. The tests also confirm our anticipation that the greater the number of features per input module, the more advantageous is the usage of modular neural networks in the case of missing features. One possible application of this approach is in face recognition, when certain parts of a face image (like nose or eyes) are not available in some images.

For further improvement of the algorithm, different schemes can be used to compute the local and/or global error factor in the ALOPEX optimization and a more reliable algorithm for adjusting the noise with respect to the global error.

As stated earlier, one type of modular neural network is a multilayer perceptron that is not fully connected. However, just deleting random connections does not make a modular neural network. In the book *Neural Networks—A Comprehensive Foundation* (1), Simon Haykin defines a modular neural network as follows:

> A neural network is said to be modular if the computation performed by the network can be decomposed into two or more modules (subsystems) that operate on distinct inputs without communicating with each other. The outputs of the modules are mediated by an integrating unit that is not permitted to feed information back to the modules. In particular, the integrating unit both (1) decides how the outputs of the modules should be combined to form the final output of the system, and (2) decides which modules should learn which training patterns.

The idea of modular neural networks is analogous to biological systems (1,2). Our brain has many different subsystems that process sensory inputs and then feed these results to other central processing neurons in the brain. For instance, consider a person who meets someone they have not seen for a long time. To remember the identity of this person, multiple sensory inputs may be processed. Foremost perhaps is the sense of sight whereby one processes what the person looks like. That may not be enough to recognize the person because the person may have changed over the course of a number of years. However, the person's looks coupled with the person's voice, the sensory input from the ears, may be enough to provide an identity. If those two are not enough, perhaps the person wears a distinctive cologne or perfume that the olfactory senses will process and add as input to the central processing. In addition, the sense of touch may also provide more information if the person has a firm handshake or soft hands. In this way our biological system makes many different observations each processed first by some module and then the results are sent to be further processed at a central location. Indeed, there may be several layers of processing before a final result is achieved.

In addition to different modules that process the input, the same sensor may process the input in two different ways. For example, the ears process the sound of a person's voice. The pitch, tonality, volume, and speed of a person's voice are all taken into account in identifying someone. However, perhaps more important is what that person says. For instance, the person may tell you their name, a piece of data that is highly critical to identification. These data are passed to the central processing to be used to match that name with the database of people's names that one has previously met. It is easy to postulate that what someone says is processed differently and perhaps feeds to a different module in the next layer than how they say it, even though the same raw data are used.

Although the concept of a modular neural network is based on biological phenomena, it also makes sense from a purely practical viewpoint. Many real-world problems have a large amount of data points. Using this large number of points as input to a fully connected multilayer perceptron results in a very large number of weights. Just blindly trying to train a network with this approach most often results in poor performance of the network, not to mention long training times because of slow convergence (3). Sometimes there are feature extraction methods, which reduce the number of data points. However, there are times when even then the amount of data is large. Because it is desirable to have the minimum number of weights that yield good performance, a modular neural network may be a good solution. Each module is effectively able to compress its data and extract subfeatures which then are used as input to a fully connected neural network. Without this modularity, the number of weights in the network would be far greater.

APPENDIX 1

Step 1: Initializing the Network

During this step all required actions are taking place to prepare the network object for training. If the network is not trained, then the following algorithm is executed.

1. Get a name of a file containing the training set of templates.
2. Read templates into the object, get the number of modules, the number of levels, the number of modules per level in the network, the number of features per input module.
3. Add required number of modules to the network object. Assign the level numbers to each module. Set up the output table of each module.
4. For each module connect it to all modules at the next level.
5. For each module add three layers to its layer container. For each layer add nodes using the following rule:

If this is the first layer of the input module, the number of nodes equals the number of features that this module receives. If this is not an input module, the number of input nodes equals the number of templates in the training set times the number of modules from which this module receives its inputs.

If this is the output layer of a module, the number of nodes equals the number of templates in the training set.

For any intermediate layer (i), the number of nodes is computed using the following equation:

$$x_i = x_0 + [(x_n - x_0)/N) \cdot i$$

where N is the total number of layers, and x_0, x_i, x_n are the number of nodes in the first, ith, and last layers.

6. For each node in each module, add connections such that each node gets connected to all nodes in the following layer. Assign a random weight to each newly added connection. If lateral inhibition is on, also add connections between current node and all (or limited) nodes at the current layer. The weight of the lateral connection is a Gaussian function of the distance between the current node and the affected node.

Step 2: Training the Network

The training process relies on a modification of the ALOPEX algorithm, which was originally developed by Tzanakou, Michalak, and Harth for receptive field mapping in the visual pathway of frogs. In this paper, we use the following scheme for implementation of the optimization procedure:

$$W(n) = W(n-1) + \gamma \cdot \Delta W(n) \cdot E \cdot k + r(n)$$

where:

$W(n)$ = new value of the connection's weight
$W(n-1)$ = old value of the connection's weight
γ = a function of the global error value
$\Delta W(n)$ = the difference $[W(n-1) - W(n-2)]$
E = global error value
k = a constant equal to -1 if $E(n-1) - E(n-2) \geq 0$, and $+1$ otherwise. This makes sense because we would like the change of the weight to decrease when the error decreases (Table 1).
$r(n)$ = added Gaussian noise used to prevent the process from freezing at the local minimum/maximum.

We define the module global error term (E) as the summation of all local errors for each template:

$$E = \sum_{i=0}^{m} E_i$$

We use the error term as opposed to the ΔE in the traditional approach, because we want the global error to have a greater impact on the change of weights when the error value is big. As the global error becomes smaller, it will lessen the effect on the change of the local weight correspondingly. The γ parameter will modulate the change of weights being a function of error.

A local error of a template is the summation of the absolute differences between the desired and actual values of the module's output nodes to a given template. We use three different approaches for computing the local error E_i'.

$$E_i' = \left| Out_i^{desired} - Out_i^{observed} \right|$$

If $E_i' >$ threshold, then
if the desired output $Out_i^{desired}$ is 1, we set $E_i' = \exp(2 \cdot E_i') - 1$,
(This is done because we would like the values on the diagonal of the output matrix to have an increased rate of convergence.)
otherwise $E_i' = \exp(E_i') - 1$,

Table 1

ΔW	E	ΔE	k	$\Delta W \cdot E \cdot k$	W(new)
>0	↑	>0	−1	<0	decreased
>0	↓	<0	1	>0	increased
<0	↑	>0	−1	<0	decreased
<0	↓	<0	1	>0	increased

Line 1: An increase of W results in an increase of E; hence W will decrease. Line 2: An increase of W results in a decrease of E; hence W will increase. Line 3: A decrease of W results in an increase of E; hence W will decrease. Line 4: A decrease of W results in a decrease of E; hence W will increase.

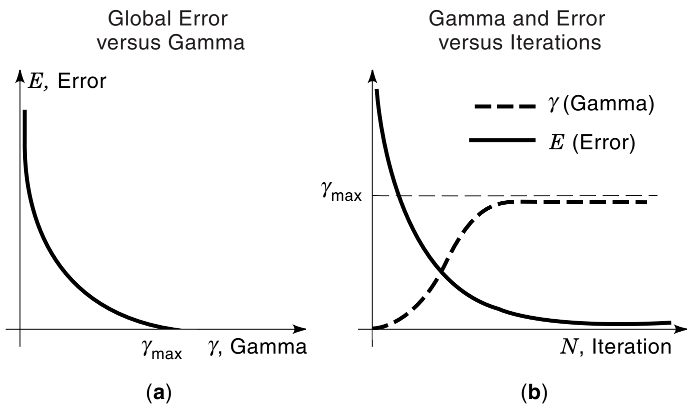


Figure 5. The graph in part (a) shows the relationship between the global error E and the value of gamma (γ). The parameter γ serves as a modulator of the term $\gamma \cdot \Delta W(n) \cdot E \cdot k$ in the optimization algorithm. The graph in part (b) shows the ideal relationship of gamma and error values with respect to the number of iterations.

or else
$E_i' = (E_i')^2$.

Figure 5 shows the relationships between the global error versus γ, on the one hand, and γ and the error curves versus the iteration number, on the other. The noise term is also adjusted accordingly so that the increase of the noise factor can become more sensitive to the change of error as the error gets smaller.

During the training cycle and for time saving purposes, we do not need to wait until the modules A . . . N, from which a module X_i is receiving its input, are trained before we start training the current module X_j. As soon as a module gets trained, it sets one of its data member flags. Therefore, step 1 involves just checking for all modules having set their trained flag. Step 2 involves additional processing.

Executing the Next Iteration

1. For each template T_i in the training set, proceed with step 2.
2. For each level L_i in the network's hierarchy, starting with an input level, and until the output level is reached, compute the output of each module belonging to the level L_i and store that output in the module's output table.
3. For each module, update weights of all connections between its nodes based on the output computed in step 2.

Computing a Module's Output

Set up the input of a module.
For each layer l_i of a module, proceed with the following two steps.
If lateral inhibition is on, for each node N_j in the level l_i, propagate the node's charge through all its lateral connections, adding any resulting charge to the "postsynaptic" node's buffer.
For each node N_m, add the content of its buffer (updated by lateral inhibition) to its charge V_k. Propagate the sig-

nal V_k through each connection C_j of the current node N_m. That is to say, add a product $V_k \cdot C_j^{weight}$ to a buffer of the "postsynaptic" node. Once the signal is propagated through all connections, set the current charge of the node N_m, as well as the value of its buffer to 0.

When the last layer is processed, store the charges accumulated in the nodes of the last layer in the module's output table.

Setting Up the Input

If a module is located at the input level of a network, obtain the input values for the charges of nodes in its input layer directly from a template T_i.

For any other module M_c, its input is obtained from output tables of the modules from which the current module M_c gets its input. (*Note:* Since each module maintains a map of its connections to other modules, we can determine which modules are connected to the current module M_c).

Updating Weights

Compute a module's global error E and determine the sign (k) of ΔE.

For each layer L_i in the current module, proceed with the following step.

For each node N_m of the layer L_i, proceed with the following step.

For each connection C_j of the node N_m add the following term to its weight:

$$r(n) + \gamma \cdot \Delta W(n) \cdot E \cdot k$$

(*Note:* The procedures for computing the global error after adjusting the noise and the parameter γ are described in "Training the Network.")

Step 3: Testing the Network

Once the network is fully trained (that is to say, the global error of each module satisfies a set threshold), we can proceed with testing. At this stage we obtain the input templates from a file and apply them to the network object. For each template, we repeat the following steps:

1. Get input values from template.
2. Compute output of a network (in the same order as during training).
3. Output results.
4. With next template do steps 1–4.

BIBLIOGRAPHY

1. S. Haykin, *Neural Networks—A Comprehensive Foundation,* New York: Macmillan, 1994.
2. T. Hrycej, *Modular Learning in Neural Networks,* New York: Wiley, 1992.
3. C. Rodriguez et al., A modular neural network approach to fault diagnosis, *IEEE Trans. Neural Netw.,* **7**: 326–340, 1996.

Reading List

S. Deutsch and E. Micheli-Tzanakou, *Neuro-Electric Systems,* New York: New York Univ. Press, 1987.

J. A. Freeman and D M. Skapura, *Neural Networks; Algorithms, Applications, and Programming Techniques,* Chap. 1, Reading, MA: Addison-Wesley, 1998.

E. Harth and E. Tzanakou, ALOPEX: A stochastic method for determining Visual Receptive Fields, *Vision Res.,* **14**: 1475–1482, 1974.

R. Hecht-Nielsen, *Neurocomputing,* Reading, MA: Addison-Wesley, 1990.

G. Held, *Data Compression,* New York: Wiley, 1987.

F. Hlawatsch and G. Boudreaux-Bartels, Linear and Quadratic Time-Frequency Signal Representations, *IEEE Signal Process. Mag.,* **9** (2): 21–67, 1992.

M. Hu, Visual pattern recognition by moment invariants, *IRE Trans. Inf. Theory,* **8**: 179–187, 1962.

R. Lippmann, An introduction to computing with neural networks, *IEEE Acoust. Speech Signal Process. Mag.,* **4** (2): 4–22, 1987.

K. Mehrotra, C. Mohan, and S. Ranka, Bounds on the number of samples needed for neural learning, *IEEE Trans. Neural Netw.,* **2**: 548–558, 1991.

L. Melissaratos and E. Micheli-Tzanakou, A parallel implementation of the ALOPEX process, *J. Med. Syst.,* **13** (5): 243–252, 1989.

E. Micheli-Tzanakou, Neural networks in biomedical signal processing, in J. Bronzino (ed.), *The Biomedical Engineering Handbook,* Boca Raton, FL: CRC Press, 1995, Chap. 60, pp. 917–932.

E. Micheli-Tzanakou, et al., Comparison of neural network algorithms for face recognition, *Simulation,* **64** (1): 15–27, 1995.

Y. Shang and B. Wah, Global optimization for neural network training, *Computer,* **29** (3): 45–54, 1996.

E. Tzanakou, R. Michalak, and E. Harth, The ALOPEX process: Visual receptive fields with response feedback, *Biol. Bybern,* **35**: 161–174, 1979.

P. Wasserman, *Advanced Methods in Neural Computing,* New York: Van Nostrand-Reinhold, 1993.

P. Wasserman, *Neural Computing: Theory and Practice,* New York: Van Nostrand-Reinhold, 1989.

D. Zahner and E. Micheli-Tzanakou, Artificial neural networks: Definitions, methods, applications, in J. Bronzino (ed.), *The Biomedical Engineering Handbook,* Boca Raton, FL: CRC Press, 1995, Chap. 184, pp. 2699–2715.

EVANGELIA MICHELI-TZANAKOU
SERGEY ALEYNIKOV
Rutgers University

NEURAL NETS, FEEDFORWARD.

See FEEDFORWARD NEURAL NETS.