129: Artificial Neural Networks

Ajith Abraham
Oklahoma State University, Stillwater, OK, USA

1 INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS

A general introduction to artificial intelligence methods of measuring signal processing is given in **Article 128, Nature and Scope of AI Techniques, Volume 2**.

The human brain provides proof of the existence of massive neural networks that can succeed at those cognitive, perceptual, and control tasks in which humans are successful. The brain is capable of computationally demanding perceptual acts (e.g. recognition of faces, speech) and control activities (e.g. body movements and body functions). The advantage of the brain is its effective use of massive parallelism, the highly parallel computing structure, and the imprecise information-processing capability. The human brain is a collection of more than 10 billion interconnected neurons. Each neuron is a cell (Figure 1) that uses biochemical reactions to receive, process, and transmit information.

Treelike networks of nerve fibers called *dendrites* are connected to the cell body or soma, where the cell nucleus is located. Extending from the cell body is a single long fiber called the *axon*, which eventually branches into strands and substrands, and are connected to other neurons through synaptic terminals or synapses.

The transmission of signals from one neuron to another at synapses is a complex chemical process in which specific transmitter substances are released from the sending end of the junction. The effect is to raise or lower the electrical potential inside the body of the receiving cell. If the potential reaches a threshold, a pulse is sent down the axon and the cell is 'fired'.

Artificial neural networks (ANN) have been developed as generalizations of mathematical models of biological nervous systems. A first wave of interest in neural networks (also known as *connectionist models* or *parallel distributed processing*) emerged after the introduction of simplified neurons by McCulloch and Pitts (1943).

The basic processing elements of neural networks are called *artificial neurons*, or *simply neurons* or *nodes*. In a simplified mathematical model of the neuron, the effects of the synapses are represented by connection weights that modulate the effect of the associated input signals, and the nonlinear characteristic exhibited by neurons is represented by a transfer function. The neuron impulse is then computed as the weighted sum of the input signals, transformed by the transfer function. The learning capability of an artificial neuron is achieved by adjusting the weights in accordance to the chosen learning algorithm.

Handbook of Measuring System Design, edited by Peter H. Sydenham and Richard Thorn.
 ISBN: 0-470-02143-8.

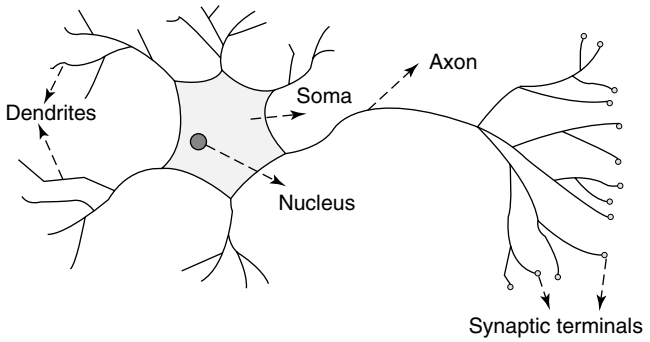

Figure 1. Mammalian neuron.

A typical artificial neuron and the modeling of a multilayered neural network are illustrated in Figure 2. Referring to Figure 2, the signal flow from inputs $x_1, \ldots, x_n$ is considered to be unidirectional, which are indicated by arrows, as is a neuron's output signal flow (O). The neuron output signal O is given by the following relationship:

$$O = f(net) = f\left(\sum_{j=1}^{n} w_j x_j\right) \tag{1}$$

where w_j is the weight vector, and the function *f(net)* is referred to as an activation (transfer) function. The variable *net* is defined as a scalar product of the weight and input vectors,

$$net = w^{\mathrm{T}} x = w_1 x_1 + \cdots + w_n x_n \tag{2}$$

where T is the transpose of a matrix, and, in the simplest case, the output value O is computed as

$$O = f(net) = \begin{cases} 1 & if\ w^{\mathrm{T}} x \geq \theta \\ 0 & otherwise \end{cases} \tag{3}$$

where θ is called the threshold level; and this type of node is called a *linear threshold unit*.

2 NEURAL NETWORK ARCHITECTURES

The basic architecture consists of three types of neuron layers: input, hidden, and output layers. In feed-forward networks, the signal flow is from input to output units, strictly in a feed-forward direction. The data processing can extend over multiple (layers of) units, but no feedback connections are present. Recurrent networks contain feedback connections. Contrary to feed-forward networks, the dynamical properties of the network are important. In some cases, the activation values of the units undergo a relaxation process such that the network will evolve to a stable state in which these activations do not change anymore. In other applications, the changes of the activation values of the output neurons are significant, such that the dynamical behavior constitutes the output of the network. There are several other neural network architectures (Elman network, adaptive resonance theory maps, competitive networks, etc.), depending on the properties and requirement of the application. The reader can refer to Bishop (1995) for an extensive overview of the different neural network architectures and learning algorithms.

A neural network has to be configured such that the application of a set of inputs produces the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to train the neural network by feeding it teaching patterns and letting it change its weights according to some learning rule. The learning situations in neural networks may be classified into three distinct sorts. These are supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, an input vector is presented at the inputs together with a set of desired responses, one for each node, at the output layer. A forward pass is done, and the errors or discrepancies between the desired and actual response for each node in the output layer are found. These are then used to determine weight changes in the net according to the prevailing learning rule. The term *supervised* originates from the fact that the desired signals on individual output nodes are provided by an external teacher.

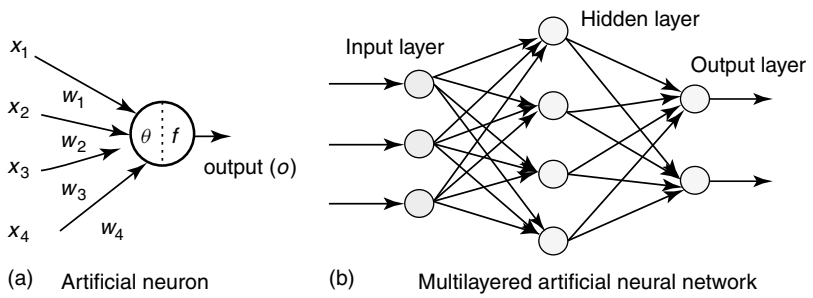

Figure 2. Architecture of an artificial neuron and a multilayered neural network.

The best-known examples of this technique occur in the backpropagation algorithm, the delta rule, and the perceptron rule. In unsupervised learning (or self-organization), a (output) unit is trained to respond to clusters of pattern within the input. In this paradigm, the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classified; rather, the system must develop its own representation of the input stimuli. Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. These two characteristics, trial-and-error search and delayed reward are the two most important distinguishing features of reinforcement learning.

3 NEURAL NETWORK LEARNING

3.1 Hebbian learning

The learning paradigms discussed above result in an adjustment of the weights of the connections between units, according to some modification rule. Perhaps the most influential work in connectionism's history is the contribution of Hebb (1949), where he presented a theory of behavior based, as much as possible, on the physiology of the nervous system.

The most important concept to emerge from Hebb's work was his formal statement (known as *Hebb's postulate*) of how learning could occur. Learning was based on the modification of synaptic connections between neurons. Specifically, when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased. The principles underlying this statement have become known as *Hebbian Learning*. Virtually, most of the neural network learning techniques can be considered as a variant of the Hebbian learning rule. The basic idea is that if two neurons are active simultaneously, their interconnection must be strengthened. If we consider a single layer net, one of the interconnected neurons will be an input unit and one an output unit. If the data are represented in bipolar form, it is easy to express the desired weight update as

$$w_i(new) = w_i(old) + x_i o,$$

where o is the desired output for

$$i = 1 \text{ to } n(\text{inputs}).$$

Unfortunately, plain Hebbian learning continually strengthens its weights without bound (unless the input data is properly normalized).

3.2 Perceptron learning rule

The perceptron is a single layer neural network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector. The training technique used is called the *perceptron-learning rule*. Perceptrons are especially suited for simple problems in pattern classification.

Suppose we have a set of learning samples consisting of an input vector x and a desired output $d(k)$. For a classification task, the $d(k)$ is usually $+1$ or -1. The perceptron-learning rule is very simple and can be stated as follows:

1. Start with random weights for the connections.
2. Select an input vector x from the set of training samples.
3. If output $y_k \neq d(k)$ (the perceptron gives an incorrect response), modify all connections w_i according to: $\delta w_i = \eta(d_k - y_k)x_i$; ($\eta =$ learning rate).
4. Go back to step 2.

Note that the procedure is very similar to the Hebb rule; the only difference is that when the network responds correctly, no connection weights are modified.

4 BACKPROPAGATION LEARNING

The simple perceptron is just able to handle linearly separable or linearly independent problems. By taking the partial derivative of the error of the network with respect to each weight, we will learn a little about the direction the error of the network is moving.

In fact, if we take the negative of this derivative (i.e. the rate change of the error as the value of the weight increases) and then proceed to add it to the weight, the error will decrease until it reaches a local minima. This makes sense because if the derivative is positive, this tells us that the error is increasing when the weight is increasing. The obvious thing to do then is to add a negative value to the weight and vice versa if the derivative is negative. Because the taking of these partial derivatives and then applying them to each of the weights takes place, starting from the output layer to hidden layer weights, then the hidden layer to input layer weights (as it turns out, this is necessary since

changing these set of weights requires that we know the partial derivatives calculated in the layer downstream), this algorithm has been called the *backpropagation algorithm*.

A neural network can be trained in two different modes: online and batch modes. The number of weight updates of the two methods for the same number of data presentations is very different.

The online method weight updates are computed for each input data sample, and the weights are modified after each sample.

An alternative solution is to compute the weight update for each input sample, but store these values during one pass through the training set which is called an *epoch*.

At the end of the epoch, all the contributions are added, and only then the weights will be updated with the composite value. This method adapts the weights with a cumulative weight update, so it will follow the gradient more closely. It is called the *batch-training mode*.

Training basically involves feeding training samples as input vectors through a neural network, calculating the error of the output layer, and then adjusting the weights of the network to minimize the error.

The average of all the squared errors (E) for the outputs is computed to make the derivative easier. Once the error is computed, the weights can be updated one by one. In the batched mode variant, the descent is based on the gradient ∇E for the total training set

$$\Delta w_{ij}(n) = -\eta^* \frac{\delta E}{\delta w_{ij}} + \alpha^* \Delta w_{ij}(n-1) \qquad (4)$$

where η and α are the learning rate and momentum respectively.

The momentum term determines the effect of past weight changes on the current direction of movement in the weight space. A good choice of both η and α are required for the training success and the speed of the neural-network learning.

It has been proven that backpropagation learning with sufficient hidden layers can approximate any nonlinear function to arbitrary accuracy. This makes backpropagation learning neural network a good candidate for signal prediction and system modeling.

5 TRAINING AND TESTING NEURAL NETWORKS

The best training procedure is to compile a wide range of examples (for more complex problems, more examples are required), which exhibit all the different characteristics of the problem.

To create a robust and reliable network, in some cases, some noise or other randomness is added to the training data to get the network familiarized with noise and natural variability in real data.

Poor training data inevitably leads to an unreliable and unpredictable network. Usually, the network is trained for a prefixed number of epochs or when the output error decreases below a particular error threshold.

Special care is to be taken not to overtrain the network. By overtraining, the network may become too adapted in learning the samples from the training set, and thus may be unable to accurately classify samples outside of the training set.

Figure 3 illustrates the classification results of an overtrained network. The task is to correctly classify two patterns X and Y. Training patterns are shown by '●' and test patterns by '♣'. The test patterns were not shown during the training phase.

As shown in Figure 3 (left side), each class of test data has been classified correctly, even though they were not seen during training. The trained network is said to have good generalization performance. Figure 3 (right side) illustrates some misclassification of the test data. The network initially learns to detect the global features of the input and, as a consequence, generalizes very well. But after prolonged training, the network starts to recognize individual input/output pairs rather than settling for weights that generally describe the mapping for the whole training set (Fausett, 1994).

5.1 Choosing the number of neurons

The number of hidden neurons affects how well the network is able to separate the data. A large number of hidden neurons will ensure correct learning, and the network is able to correctly predict the data it has been trained on, but its performance on new data, its ability to generalize, is compromised. With too few hidden neurons, the network may be unable to learn the relationships amongst the data and the error will fail to fall below an acceptable level. Thus, selection of the number of hidden neurons is a crucial decision.

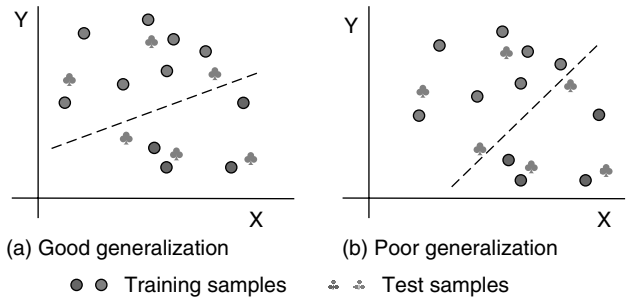

Figure 3. Illustration of generalization performance.

5.2 Choosing the initial weights

The learning algorithm uses a steepest descent technique, which rolls straight downhill in weight space until the first valley is reached. This makes the choice of initial starting point in the multidimensional weight space critical. However, there are no recommended rules for this selection except trying several different starting weight values to see if the network results are improved.

5.3 Choosing the learning rate

Learning rate effectively controls the size of the step that is taken in multidimensional weight space when each weight is modified. If the selected learning rate is too large, then the local minimum may be overstepped constantly, resulting in oscillations and slow convergence to the lower error state. If the learning rate is too low, the number of iterations required may be too large, resulting in slow performance.

6 HIGHER ORDER LEARNING ALGORITHMS

Backpropagation (BP) often gets stuck at a local minimum mainly because of the random initialization of weights. For some initial weight settings, BP may not be able to reach a global minimum of weight space, while for other initializations the same network is able to reach an optimal minimum.

A long recognized bane of analysis of the error surface and the performance of training algorithms is the presence of multiple stationary points, including multiple minima.

Empirical experience with training algorithms show that different initialization of weights yield different resulting networks. Hence, multiple minima not only exist, but there may be huge numbers of them.

In practice, there are four types of optimization algorithms that are used to optimize the weights. The first three methods, gradient descent, conjugate gradients, and quasi-Newton, are general optimization methods whose operation can be understood in the context of minimization of a quadratic error function.

Although the error surface is surely not quadratic, for differentiable node functions, it will be so in a sufficiently small neighborhood of a local minimum, and such an analysis provides information about the behavior of the training algorithm over the span of a few iterations and also as it approaches its goal.

The fourth method of Levenberg and Marquardt is specifically adapted to the minimization of an error function that arises from a squared error criterion of the form we are assuming. A common feature of these training algorithms is the requirement of repeated efficient calculation of gradients. The reader can refer to Bishop (1995) for an extensive coverage of higher-order learning algorithms.

Even though artificial neural networks are capable of performing a wide variety of tasks, in practice, sometimes, they deliver only marginal performance. Inappropriate topology selection and learning algorithm are frequently blamed. There is little reason to expect that one can find a uniformly best algorithm for selecting the weights in a feedforward artificial neural network. This is in accordance with the no free lunch theorem, which explains that for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class (Macready and Wolpert, 1997).

The design of artificial neural networks using evolutionary algorithms has been widely explored. Evolutionary algorithms are used to adapt the connection weights, network architecture, and so on, according to the problem environment.

A distinct feature of evolutionary neural networks is their adaptability to a dynamic environment. In other words, such neural networks can adapt to an environment as well as changes in the environment. The two forms of adaptation, evolution and learning in evolutionary artificial neural networks, make their adaptation to a dynamic environment much more effective and efficient than the conventional learning approach. Refer to Abraham (2004) for more technical information related to evolutionary design of neural networks.

7 DESIGNING ARTIFICIAL NEURAL NETWORKS

To illustrate the design of artificial neural networks, the Mackey-Glass chaotic time series (Box and Jenkins, 1970) benchmark is used. The performance of the designed neural network is evaluated for different architectures and activation functions. The Mackey-Glass differential equation is a chaotic time series for some values of the parameters $x(0)$ and τ.

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1\ x(t). \tag{5}$$

We used the value $x(t-18)$, $x(t-12)$, $x(t-6)$, $x(t)$ to predict $x(t+6)$. Fourth order Runge-Kutta method was used to generate 1000 data series. The time step used in the method is 0.1 and initial condition were $x(0) = 1.2$, $\tau =$

Table 1. Training and test performance for Mackey-Glass Series for different architectures.

Hidden neurons	Root mean-squared error	
	Training data	Test data
14	0.0890	0.0880
16	0.0824	0.0860
18	0.0764	0.0750
20	0.0452	0.0442
24	0.0439	0.0437

17, $x(t) = 0$ for $t < 0$. The first 500 data sets were used for training and remaining data for testing.

7.1 Network architecture

A feed-forward neural network with four input neurons, one hidden layer and one output neuron is used. Weights were randomly initialized and the learning rate and momentum are set at 0.05 and 0.1 respectively. The numbers of hidden neurons are varied (14, 16, 18, 20, 24) and the generalization performance is reported in Table 1. All networks were trained for an identical number of stochastic updates (2500 epochs).

7.2 Role of activation functions

The effect of two different node activation functions in the hidden layer, log-sigmoidal activation function LSAF and tanh-sigmoidal activation function TSAF), keeping 24 hidden neurons for the backpropagation learning algorithm, is illustrated in Figure 4. Table 2 summarizes the empirical results for training and generalization for the two node transfer functions. The generalization looks better with TSAF.

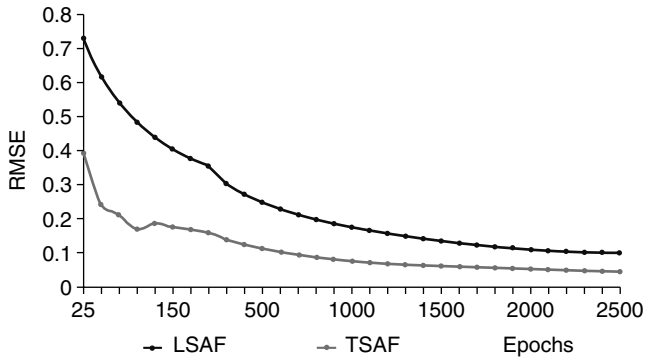

Figure 4. Convergence of training for different node transfer function.

Table 2. Mackey-Glass time series: training and generalization performance for different activation functions.

Activation function	Root mean-squared error	
	Training	Test
TSAF	0.0439	0.0437
LSAF	0.0970	0.0950

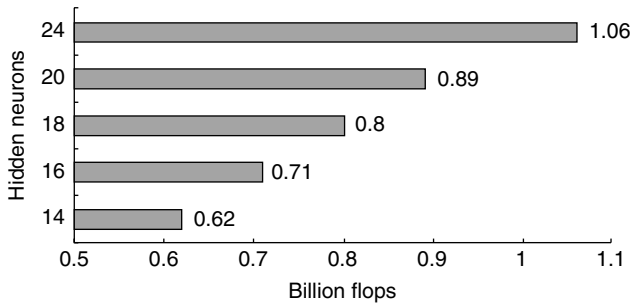

Figure 5. Computational complexity for different architectures.

Figure 5 illustrates the computational complexity in billion flops for different numbers of hidden neurons. At present, neural network design relies heavily on human experts who have sufficient knowledge about the different aspects of the network and the problem domain. As the complexity of the problem domain increases, manual design becomes more difficult.

8 SELF-ORGANIZING FEATURE MAP AND RADIAL BASIS FUNCTION NETWORK

8.1 Self-organizing feature map

Self-organizing Feature Maps SOFM is a data visualization technique proposed by Kohonen (1988), which reduces the dimensions of data through the use of self-organizing neural networks.

A SOFM learns the categorization, topology, and distribution of input vectors. SOFM allocate more neurons to recognize parts of the input space where many input vectors occur and allocate fewer neurons to parts of the input space where few input vectors occur. Neurons next to each other in the network learn to respond to similar vectors.

SOFM can learn to detect regularities and correlations in their input and adapt their future responses to that input accordingly. An important feature of the SOFM learning

algorithm is that it allows neurons that are neighbors to the winning neuron to be output values. Thus, the transition of output vectors is much smoother than that obtained with competitive layers, where only one neuron has an output at a time.

The problem that data visualization attempts to solve is that humans simply cannot visualize high-dimensional data. The way SOFM goes about reducing dimensions is by producing a map of usually 1 or 2 dimensions, which plot the similarities of the data by grouping similar data items together (data clustering). In this process, SOFM accomplish two things, they reduce dimensions and display similarities.

It is important to note that while a self-organizing map does not take long to organize itself so that neighboring neurons recognize similar inputs, it can take a long time for the map to finally arrange itself according to the distribution of input vectors.

8.2 Radial basis function network

The Radial Basis Function (RBF) network is a three-layer feed-forward network that uses a linear transfer function for the output units and a nonlinear transfer function (normally the Gaussian) for the hidden layer neurons (Chen, Cowan and Grant, 1991). Radial basis networks may require more neurons than standard feed-forward backpropagation networks, but often they can be designed with lesser time. They perform well when many training data are available.

Much of the inspiration for RBF networks has come from traditional statistical pattern classification techniques. The input layer is simply a fan-out layer and does no processing. The second or hidden layer performs a nonlinear mapping from the input space into a (usually) higher dimensional space whose activation function is selected from a class of functions called basis functions.

The final layer performs a simple weighted sum with a linear output. Contrary to BP networks, the weights of the hidden layer basis units (input to hidden layer) are set using some clustering techniques. The idea is that the patterns in the input space form clusters. If the centers of these clusters are known, then the Euclidean distance from the cluster center can be measured. As the input data moves away from the connection weights, the activation value reduces. This distance measure is made nonlinear in such a way that for input data close to a cluster center gets a value close to 1. Once the hidden layer weights are set, a second phase of training (usually backpropagation) is used to adjust the output weights.

9 RECURRENT NEURAL NETWORKS AND ADAPTIVE RESONANCE THEORY

9.1 Recurrent neural networks

Recurrent networks are the state of the art in nonlinear time series prediction, system identification, and temporal pattern classification. As the output of the network at time t is used along with a new input to compute the output of the network at time $t + 1$, the response of the network is dynamic (Mandic and Chambers, 2001).

Time Lag Recurrent Networks (TLRN) are multilayered perceptrons extended with short-term memory structures that have local recurrent connections. The recurrent neural network is a very appropriate model for processing temporal (time-varying) information.

Examples of temporal problems include time-series prediction, system identification, and temporal pattern recognition. A simple recurrent neural network could be constructed by a modification of the multilayered feed-forward network with the addition of a 'context layer'. The context layer is added to the structure, which retains information between observations. At each time step, new inputs are fed to the network. The previous contents of the hidden layer are passed into the context layer. These then feed back into the hidden layer in the next time step. Initially, the context layer contains nothing, so the output from the hidden layer after the first input to the network will be the same as if there is no context layer. Weights are calculated in the same way for the new connections from and to the context layer from the hidden layer.

The training algorithm used in TLRN (backpropagation through time) is more advanced than standard backpropagation algorithm. Very often, TLRN requires a smaller network to learn temporal problems when compared to MLP that use extra inputs to represent the past samples. TLRN is biologically more plausible and computationally more powerful than other adaptive models such as the hidden Markov model.

Some popular recurrent network architectures are the Elman recurrent network in which the hidden unit activation values are fed back to an extra set of input units and the Jordan recurrent network in which output values are fed back into hidden units.

9.2 Adaptive resonance theory

Adaptive Resonance Theory (ART) was initially introduced by Grossberg (1976) as a theory of human information processing. ART neural networks are extensively used for

supervised and unsupervised classification tasks and function approximation.

There exist many different variations of ART networks today (Carpenter and Grossberg, 1998). For example, ART1 performs unsupervised learning for binary input patterns, ART2 is modified to handle both analog and binary input patterns, and ART3 performs parallel searches of distributed recognition codes in a multilevel network hierarchy. Fuzzy ARTMAP represents a synthesis of elements from neural networks, expert systems, and fuzzy logic.

10 SUMMARY

This section presented the biological motivation and fundamental aspects of modeling artificial neural networks. Performance of feed-forward artificial neural networks for a function approximation problem is demonstrated. Advantages of some specific neural network architectures and learning algorithms are also discussed.

REFERENCES

Abraham, A. (2004) Meta-Learning Evolutionary Artificial Neural Networks, *Neurocomputing Journal*, Vol. 56c, Elsevier Science, Netherlands, (1–38).

Bishop, C.M. (1995) *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, UK.

Box, G.E.P. and Jenkins, G.M. (1970) *Time Series Analysis, Forecasting and Control*, Holden Day, San Francisco, CA.

Carpenter, G. and Grossberg, S. (1998) in *Adaptive Resonance Theory (ART), The Handbook of Brain Theory and Neural Networks*, (ed. M.A. Arbib), MIT Press, Cambridge, MA, (pp. 79–82).

Chen, S., Cowan, C.F.N. and Grant, P.M. (1991) Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks. *IEEE Transactions on Neural Networks*, **2**(2), 302–309.

Fausett, L. (1994) *Fundamentals of Neural Networks*, Prentice Hall, USA.

Grossberg, S. (1976) Adaptive Pattern Classification and Universal Recoding: Parallel Development and Coding of Neural Feature Detectors. *Biological Cybernetics*, **23**, 121–134.

Hebb, D.O. (1949) *The Organization of Behavior*, John Wiley, New York.

Kohonen, T. (1988) *Self-Organization and Associative Memory*, Springer-Verlag, New York.

Macready, W.G. and Wolpert, D.H. (1997) The No Free Lunch Theorems. *IEEE Transactions on Evolutionary Computing*, **1**(1), 67–82.

Mandic, D. and Chambers, J. (2001) *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*, John Wiley & Sons, New York.

McCulloch, W.S. and Pitts, W.H. (1943) A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, **5**, 115–133.