ELSEVIER

Journal of Microbiological Methods 43 (2000) 3–31

Journal of Microbiological Methods

www.elsevier.com/locate/jmicmeth

Artificial neural networks: fundamentals, computing, design, and application

I.A. Basheer[a,*], M. Hajmeer[b]

[a]Engineering Service Center, The Headquarters Transportation Laboratory, CalTrans, Sacramento, CA 95819, USA
[b]Department of Animal Sciences and Industry, Kansas State University, Manhattan, KS 66506, USA

Abstract

Artificial neural networks (ANNs) are relatively new computational tools that have found extensive utilization in solving many complex real-world problems. The attractiveness of ANNs comes from their remarkable information processing characteristics pertinent mainly to nonlinearity, high parallelism, fault and noise tolerance, and learning and generalization capabilities. This paper aims to familiarize the reader with ANN-based computing (neurocomputing) and to serve as a useful companion practical guide and toolkit for the ANNs modeler along the course of ANN project development. The history of the evolution of neurocomputing and its relation to the field of neurobiology is briefly discussed. ANNs are compared to both expert systems and statistical regression and their advantages and limitations are outlined. A bird's eye review of the various types of ANNs and the related learning rules is presented, with special emphasis on backpropagation (BP) ANNs theory and design. A generalized methodology for developing successful ANNs projects from conceptualization, to design, to implementation, is described. The most common problems that BPANNs developers face during training are summarized in conjunction with possible causes and remedies. Finally, as a practical application, BPANNs were used to model the microbial growth curves of *S. flexneri*. The developed model was reasonably accurate in simulating both training and test time-dependent growth curves as affected by temperature and pH. © 2000 Published by Elsevier Science B.V.

Keywords: Artificial neural networks; Backpropagation; Growth curves; History; Modeling; *S. flexneri*

1. Introduction

Artificial Neural Networks (ANNs) are computational modeling tools that have recently emerged and found extensive acceptance in many disciplines for modeling complex real-world problems. ANNs may be defined as structures comprised of densely interconnected adaptive simple processing elements (called artificial neurons or nodes) that are capable of performing massively parallel computations for data processing and knowledge representation (Hecht-Nielsen, 1990; Schalkoff, 1997). Although ANNs are drastic abstractions of the biological counterparts, the idea of ANNs is not to replicate the operation of the biological systems but to make use of what is known about the functionality of the biological networks for solving complex problems. The attractiveness of ANNs comes from the remarkable information processing characteristics of the biological system such as nonlinearity, high parallelism, robustness, fault and failure tolerance, learning, ability to handle imprecise and fuzzy information, and their capability to generalize (Jain et al., 1996). Artificial models possessing such characteristics are desirable because

*Corresponding author.
E-mail address: imad.basheer@dot.ca.gov (I.A. Basheer).

PII: S0167-7012(00)00201-3

(i) nonlinearity allows better fit to the data, (ii) noise-insensitivity provides accurate prediction in the presence of uncertain data and measurement errors, (iii) high parallelism implies fast processing and hardware failure-tolerance, (iv) learning and adaptivity allow the system to update (modify) its internal structure in response to changing environment, and (v) generalization enables application of the model to unlearned data. The main objective of ANN-based computing (neurocomputing) is to develop mathematical algorithms that will enable ANNs to learn by mimicking information processing and knowledge acquisition in the human brain.

ANN-based models are empirical in nature, however they can provide practically accurate solutions for precisely or imprecisely formulated problems and for phenomena that are only understood through experimental data and field observations. In microbiology, ANNs have been utilized in a variety of applications ranging from modeling, classification, pattern recognition, and multivariate data analysis. Sample applications include (i) interpreting pyrolysis mass spectrometry, GC, and HPLC data, (ii) pattern recognition of DNA, RNA, protein structure, and microscopic images, (iii) prediction of microbial growth, biomass, and shelf life of food products, and (iv) identification of microorganisms and molecules (see Application section for references).

The objective of this paper is to provide a preliminary understanding of ANNs and answer the why and when these computational tools are needed, the motivation behind their development, and their relation to biological systems and other modeling methodologies, the various learning rules and ANN types, computations involved, design considerations, application to real-world problems, and advantages and limitations. Such understanding is essential for making efficient use of their features. More emphasis will be given to BPANNs as being the most popular and versatile type of networks.

2. ANNs and biological neural networks

Because the biological neuron is the basic building block of the nervous system, its operation will be briefly discussed for understanding artificial neuron operation and the analogy between ANNs and biological networks.

2.1. Biological neuron

The human nervous system consists of billions of neurons of various types and lengths relevant to their location in the body (Schalkoff, 1997). Fig. 1a shows a schematic of an oversimplified biological neuron with three major functional units — dendrites, cell body, and axon. The cell body has a nucleus that contains information about heredity traits, and a plasma that holds the molecular equipment used for producing the material needed by the neuron (Jain et al., 1996). The dendrites receive signals from other neurons and pass them over to the cell body. The

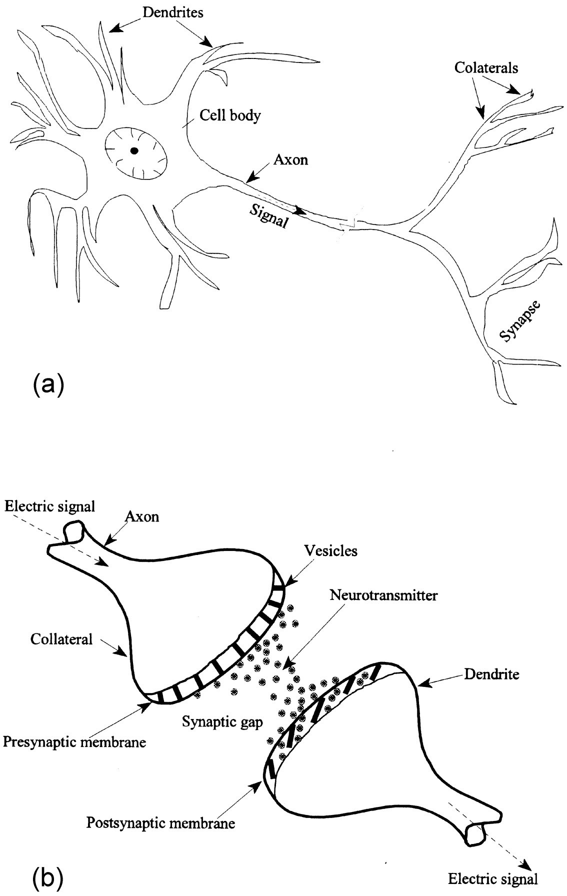


Fig. 1. (a) Schematic of biological neuron. (b) Mechanism of signal transfer between two biological neurons.

total receiving area of the dendrites of a typical neuron is approximately 0.25 mm^2 (Zupan and Gasteiger, 1993). The axon, which branches into collaterals, receives signals from the cell body and carries them away through the synapse (a microscopic gap) to the dendrites of neighboring neurons. A schematic illustration of the signal transfer between two neurons through the synapse is shown in Fig. 1b. An impulse, in the form of an electric signal, travels within the dendrites and through the cell body towards the pre-synaptic membrane of the synapse. Upon arrival at the membrane, a neurotransmitter (chemical) is released from the vesicles in quantities proportional to the strength of the incoming signal. The neurotransmitter diffuses within the synaptic gap towards the post-synaptic membrane, and eventually into the dendrites of neighboring neurons, thus forcing them (depending on the threshold of the receiving neuron) to generate a new electrical signal. The generated signal passes through the second neuron(s) in a manner identical to that just described. The amount of signal that passes through a receiving neuron depends on the intensity of the signal emanating from each of the feeding neurons, their synaptic strengths, and the threshold of the receiving neuron. Because a neuron has a large number of dendrites/synapses, it can receive and transfer many signals simultaneously. These signals may either assist (excite) or inhibit the firing of the neuron. This simplified mechanism of signal transfer constituted the fundamental step of early neurocomputing development (e.g., the binary threshold unit of McCulloh and Pitts, 1943) and the operation of the building unit of ANNs.

2.2. Analogy

The crude analogy between artificial neuron and biological neuron is that the connections between nodes represent the axons and dendrites, the connection weights represent the synapses, and the threshold approximates the activity in the soma (Jain et al., 1996). Fig. 2 illustrates *n* biological neurons with various signals of intensity *x* and synaptic strength *w* feeding into a neuron with a threshold of *b*, and the equivalent artificial neurons system. Integration of the signals in artificial neuronal sys-

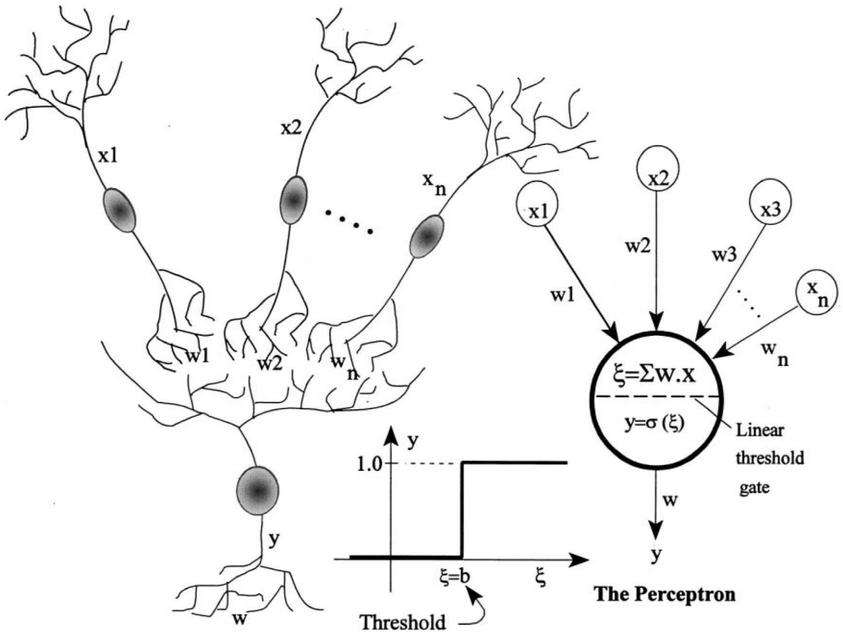


Fig. 2. Signal interaction from *n* neurons and analogy to signal summing in an artificial neuron comprising the single layer perceptron.

tems will be discussed later. Both the biological network and ANN learn by incrementally adjusting the magnitudes of the weights or synapses' strengths (Zupan and Gasteiger, 1993).

2.3. Artificial neuron

In 1958, Rosenblatt introduced the mechanics of the single artificial neuron and introduced the 'Perceptron' to solve problems in the area of character recognition (Hecht-Nielsen, 1990). Basic findings from the biological neuron operation enabled early researchers (e.g., McCulloh and Pitts, 1943) to model the operation of simple artificial neurons. An artificial processing neuron receives inputs as stimuli from the environment, combines them in a special way to form a 'net' input (ξ), passes that over through a linear threshold gate, and transmits the (output, y) signal forward to another neuron or the environment, as shown in Fig. 2. Only when ξ exceeds (i.e., is stronger than) the neuron's threshold limit (also called bias, b), will the neuron fire (i.e, becomes activated). Commonly, linear neuron dynamics are assumed for calculating ξ (Haykin, 1994). The net input is computed as the inner (dot) product of the input signals (x) impinging on the neuron and their strengths (w). For n signals, the perceptron neuron operation is expressed as

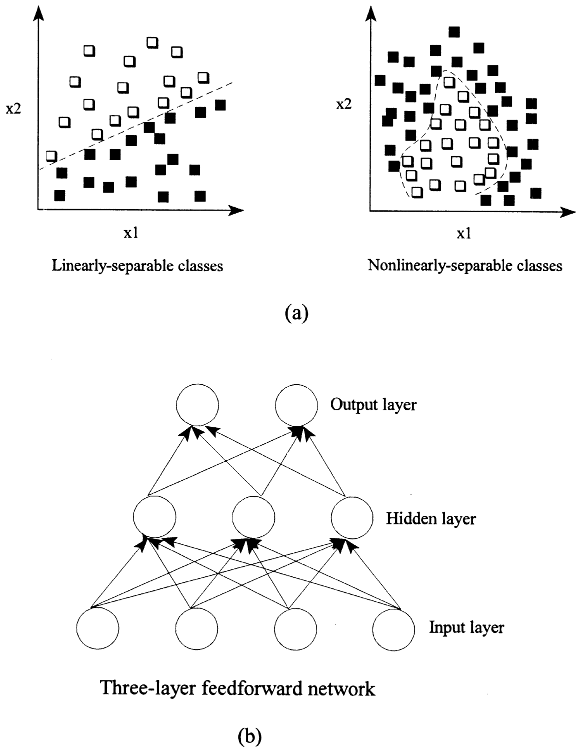
$$y = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i x_i \geq b, \\ 0, & \text{if } \sum_{i=1}^{n} w_i x_i < b, \end{cases} \tag{1}$$

with 1 indicating 'on' and 0 indicating 'off' (Fig. 2), or class A and B, respectively, in solving classification problems. Positive connection weights ($w_i > 0$) enhance the net signal (ξ) and excite the neuron, and the link is called excitory, whereas negative weights reduce ξ and inhibit the neuron activity, and the link is called inhibitory. The system comprised of an artificial neuron and the inputs as shown in Fig. 2 is called the Perceptron which establishes a mapping between the inputs activity (stimuli) and the output signal. In Eq. (1), the neuron threshold may be considered as an additional input node whose value is always unity (i.e., $x = 1$) and its connection weight is equal to b. In such case, the summation in Eq. (1) is run from 0 to n, and the net signal ξ is compared to 0.

2.4. Perceptrons

The perceptron (Fig. 2) can be trained on a set of examples using a special learning rule (Hecht-Nielsen, 1990). The perceptron weights (including the threshold) are changed in proportion to the difference (error) between the target (correct) output, Y, and the perceptron solution, y, for each example. The error is a function of all the weights and forms an irregular multidimensional complex hyperplane with many peaks, saddle points, and minima. Using a specialized search technique, the learning process strives to obtain the set of weights that corresponds to the global minimum. Rosenblatt (1962) derived the perceptron rule that will yield an optimal weight vector in a finite number of iterations, regardless of the initial values of the weights. This rule, however, can only perform accurately with linearly separable classes (Hecht-Nielsen, 1990), in which a linear hyperplane can place one class of objects on one side of the plane and the other class on the other side. Fig. 3a shows linearly and nonlinearly separable two-object classification problems.

In order to cope with nonlinearly separable problems, additional layer(s) of neurons placed between the input layer (containing input nodes) and the output neuron are needed leading to the multilayer perceptron (MLP) architecture (Hecht-Nielsen, 1990), as shown in Fig. 3b. Because these intermediate layers do not interact with the external environment, they are called hidden layers and their nodes called hidden nodes. The addition of intermediate layers revived the perceptron by extending its ability to solve nonlinear classification problems. Using similar neuron dynamics, the hidden neurons process the information received from the input nodes and pass them over to output layer. The learning of MLP is not as direct as that of the simple perceptron. For example, the backpropagation network (Rumelhart et al., 1986) is one type of MLP trained by the delta learning rule (Zupan and Gasteiger, 1993). However, the learning procedure is an extension of the simple perceptron algorithm so as to


x2
x1
Linearly-separable classes
x2
x1
Nonlinearly-separable classes
Output layer
Hidden layer
Input layer
Three-layer feedforward network

(a)

(b)

Fig. 3. (a) Linear vs. nonlinear separability. (b) Multilayer perceptron showing input, hidden, and output layers and nodes with feedforward links.

handle the weights connected to the hidden nodes (Hecht-Nielsen, 1990).

2.5. Biological vs. artificial network

Central to our biological neural network is the cerebral cortex (cerebrum) which is a 2–3 mm thick flat sheet of massively interconnected neurons with an approximate surface area of 2200 cm^2 containing about 10^{11} neurons (Jain et al., 1996). Each neuron is connected to 1000–10,000 other neurons (Schalkoff, 1997), making approximately 10^{14} to 10^{15} interconnections. In contrast, ANNs (e.g., backpropagation) typically range from 10 to as high as 10,000 neurons for the most sophisticated networks implementable on a digital computer, with a connection density ranging from five to 100 links per neuron (Wythoff, 1993). With regard to their operation and internal structure, ANNs are considered homogenous and often operate deterministically, whereas those of the human cortex are extremely heterogenous and operate in a mixture of complex deterministic and stochastic manner (Wythoff, 1993). With regard to functionality, it is not surprising to see that ANNs compare, though roughly, to biological networks as they are developed to mimic the computational properties of the brain (Schalkoff, 1997) such as adaptivity, noise (data) and fault (neurons and connections lost) tolerance.

2.6. Learning

The ability to learn is a peculiar feature pertaining to intelligent systems, biological or otherwise. In artificial systems, learning is viewed as the process of updating the internal representation of the system in response to external stimuli so that it can perform a specific task. This includes modifying the network architecture, which involves adjusting the weights of the links, pruning or creating some connection links, and/or changing the firing rules of the individual neurons (Schalkoff, 1997). ANN learning is performed iteratively as the network is presented with training examples, similar to the way we learn from experience. An ANN-based system is said to have learnt if it can (i) handle imprecise, fuzzy, noisy, and probabilistic information without noticeable adverse effect on response quality, and (ii) generalize from the tasks it has learned to unknown ones.

3. Neurocomputing history

In this section, the historical evolution of ANNs and neurocomputing is briefly presented. Anderson and Rosenfeld (1988) provide a detailed history along with a collection of the many major classic papers that affected ANNs evolution. Nelson and Illingworth (1990) divide 100 years of history into six notable phases: (1) Conception, 1890–1949; (2) Gestation and Birth, 1950s; (3) Early Infancy, late 1950s and the 1960s; (4) Stunted Growth, 1961–1981; (5) Late Infancy I, 1982–1985; and (6) Late Infancy II, 1986–present.

The era of conception includes the first development in brain studies and the understanding of brain mathematics. It is believed that the year 1890 was the beginning of the neurocomputing age in which the first work on brain activity was published by William James (Nelson and Illingworth, 1990). Many (e.g., Hecht-Nielsen, 1990) believe that real neurocomputing started in 1943 after McCulloh and Pitts (1943) paper on the ability of simple neural networks to compute arithmetic and logical functions. This era ended with the book *The Organization of Behavior* by Donald Hebb in which he presented his learning law for the biological neurons' synapses (Hebb, 1949), believed to have paved the road for the advent of neurocomputing (Hecht-Nielsen, 1990). The gestation and birth era began following the advances in hardware/software technology which made computer simulations possible and easier. In this era, the first neurocomputer (the Snark) was built and tested by Minsky at Princeton University in 1951, but it experienced many limitations (Hecht-Nielsen, 1990). This era ended by the development of the Dartmouth Artificial Intelligence (AI) research project which laid the foundations for extensive neurocomputing research (Nelson and Illingworth, 1990). The era of early infancy began with John von Neuman's work which was published a year after his death in a book entitled *The Computer and the Brain* (von Neuman, 1958). In the same year, Frank Rosenblatt at Cornell University introduced the first successful neurocomputer (the Mark I perceptron), designed for character recognition which is considered nowadays the oldest ANN hardware (Nelson and Illingworth, 1990). Although the Rosenblatt perceptron was a linear system, it was efficient in solving many problems and led to what is known as the 1960s ANNs hype. In this era, Rosenblatt also published his book *Principles of Neurodynamics* (Rosenblatt, 1962). The neurocomputing hype, however, did not last long due to a campaign led by Minsky and Pappert (1969) aimed at discrediting ANNs research to redirect funding back to AI. Minsky and Pappert published their book *Perceptrons* in 1969 in which they over exaggerated the limitations of the Rosenblatt's perceptron as being incapable of solving nonlinear classification problems, although such a limitation was already known (Hecht-Nielsen, 1990; Wythoff, 1993). Unfortunately, this campaign achieved its planned goal, and by the early 1970s many ANN researchers switched their attention back to AI, whereas a few 'stubborn' others continued their research. Hecht-Nielsen (1990) refers to this era as the 'quiet years' and the 'quiet research'. With the Rosenblatt perceptron and the other ANNs introduced by the 'quiet researchers', the field of neurocomputing gradually began to revive and the interest in neurocomputing renewed. Nelson and Illingworth (1990) list a few of the most important research studies that assisted the rebirth and revitalization of this field, notably the introduction of the Hopfield networks (Hopfield, 1984), developed for retrieval of complete images from fragments. The year 1986 is regarded a cornerstone in the ANNs recent history as Rumelhart et al. (1986) rediscovered the backpropagation learning algorithm after its initial development by Werbos (1974). The first physical sign of the revival of ANNs was the creation of the Annual IEEE International ANNs Conference in 1987, followed by the formation of the International Neural Network Society (INNS) and the publishing of the INNS Neural Network journal in 1988. It can be seen that the evolution of neurocomputing has witnessed many ups and downs, notable among which is the period of hibernation due to the perceptron's inability to handle nonlinear classification. Since 1986, many ANN societies have been formed, special journals published, and annual international conferences organized. At present, the field of neurocomputing is blossoming almost daily on both the theory and practical application fronts.

4. ANNs, statistics, and expert systems

4.1. Statistics

Cheng and Titterington (1994) and White (1989) provide excellent discussions about ANNs from a statistics perspective. While there are many differences between ANNs and statistics, there are also many similarities (Hanson, 1995; Wythoff, 1993). Some ANN models (e.g., feedforward networks with no hidden units) may be viewed as generalizations to some statistical models, some are closely related to statistical modeling (e.g., Hebbian learning), and others (e.g., Kohonen networks) appear to have no close relatives in the existing statistical literature (White, 1989).

There is an argument, however, as to whether ANNs are distinctly different from statistical regression or whether they are extensions of these old techniques (Cheng and Titterington, 1994). In multiple regression, an estimate of the dependent variable ($\hat{Y}$) is determined from $\hat{Y} = \Phi(A,X)$, where X is a vector of independent variables ($x_1,x_2,x_3,\ldots,x_n$), A is a vector of regression coefficients ($a_0,a_1,a_2,\ldots,a_n$), and Φ is a function relating X to $\hat{Y}$ determined from regression. The regression technique, based on mean squared error minimization of a linear function, requires that the mathematical form of both the regression equation and independent variables be known or assumed a priori. Since there is no best guess, there is no assurance that the obtained regression model is optimal. In ANNs (e.g., backpropagation), $\hat{Y} = \Psi(W,X)$, where W is a vector of all connection weights (and thresholds) in the network and Ψ is a complex function representing the ANN internal structure. The major difference is that, in multiple linear regression, Φ is a set of linear operators, whereas in ANNs Ψ represents a linear combination of a large number of simple nonlinear functions, such as sigmoid functions (Ni and Gunasekaran, 1998). Rumelhart et al. (1995) and White (1990) refer to the ANNs approach as one form of nonlinear statistical regression, whereas Werbos (1974) describes the backpropagation ANN as a tool superior to regression. Many others view ANNs as generalizations of classical regression or ‘super regression’ (Wythoff, 1993), in which the ANN regression is performed adaptively using nonlinear learning laws, whereas it is performed by matrix inversion in statistical regression. Hanson (1995) defines backpropagation ANN as a multivariate, nonlinear, nonparametric, stochastic approximation with dynamic feature extraction/selection, which makes them capable of learning arbitrary mapping. ‘Nonparametric’ indicates that, unlike conventional statistics, neither the functional form of the mean function nor the error distribution is prespecified. There is a great body of agreement among researchers that ANNs outperform statistical regression with regard to prediction accuracy (Masters, 1994; Sun et al., 1997). ANNs superiority increases as the dimensionality and/or nonlinearity of the problem increases, when traditional regression often fails to produce accurate approximations. Hecht-Nielsen (1990) reports that neurocomputing is now providing a breath of fresh new air to the 200+ years of old Gaussian statistical regression.

In the polynomial approach, the limitation is obvious: it may only be suited practically to one independent variable or low order polynomial (Specht, 1991). As the number of independent variables, N, increases the number of polynomial parameters grows as N^m, m being the order of the polynomial. On the other hand, in ANNs, the number of free parameters grows only linearly (i.e. as N) or quadratically (i.e. as N^2) for given m hidden units (Bishop, 1995). Additionally, Barron (1993) showed that the residual sum of square errors decreases as $O(1/m)$ in ANNs, and as $O(1/m)^{2/N}$ in polynomials, emphasizing the effectiveness of ANNs with three or more input parameters.

The choice between ANNs and statistical techniques depends on the problem to be solved. For example, ANNs outperform traditional statistics in solving perceptual problems such as vision, speech, etc. (Schalkoff, 1997). However, for modeling of data of low dimensionality or for approximating simple functions, classical techniques based on statistics should be tried first, and ANNs may then be employed if higher accuracy is needed. We recommend that the anticipated enhancement in prediction accuracy of ANN-based models be weighed against the increased complexity in their development and the lost desirable characteristics of the linear statistical models (e.g., simple differentiation, parametric sensitivity analysis, and optimization).

4.2. Expert systems

An expert system (ES) is a computer program that mimics the human reasoning process, which relies on logic, belief, rules of thumb, opinion, and experience (Plant and Stone, 1991; Fu, 1995). In rule-based ES, the knowledge and experience of the expert are coded as a series of IF–THEN rules. An inference engine traverses the rules in the stored knowledge base, draws conclusions, and provides the user with a solution (e.g., medical diagnosis). Unlike ANNs, ESs suffer from serious limitations, mainly their hypersensitivity to incomplete and noisy data (Fu, 1995).

Moreover, some human knowledge cannot be expressed explicitly in terms of rules (Hoffman, 1987). These and several other limitations (Fu, 1995) have stimulated the exploration of ANNs for data modeling (Sui, 1992).

ESs work in sequential manner, whereas ANNs are parallel data manipulators, and sometimes viewed as a reverse approach to ESs (Nelson and Illingworth, 1990). Other differences relate to information processing, connectivity, self-learning ability, fault tolerance, and relation to neurobiology (Fu, 1995). In order to capture the desirable features of both systems, ESs and ANNs are integrated into one hybrid system (Kandel and Langholz, 1992). In this system, ANNs learn the hidden rules from the examples, and the ES extracts them in explicit forms, thus roughly simulating a whole-brain operation (Nelson and Illingworth, 1990).

The decision as to whether to use ANNs, ESs, or theoretical modeling for an arbitrary problem depends primarily on the availability of both the theory explaining the underlaying phenomena and the data, as described in Fig. 4 (Rumelhart et al., 1995). For a problem with abundant data but unclear theory, ANNs can be a perfect tool. Conversely, when both the data and theory are inadequate, the human expert's opinion (solution) should be sought followed by coding this 'knowledge' into a set of rules using ES techniques. Finally, when the problem is rich in both data and theory, it may be possible to derive a physical (theoretical) model in which the data are used for model calibration and verification. Overlapping areas were added to Fig. 4 to characterize some other common scenarios. For instance, sufficient data and limited theory (i.e., a partially understood phenomenon) dictate the need for a semi-empirical model. Alternatively, when both theory and data are abundant but a physical model is hard to formulate, the modeler may resort to empirical modeling such as ANNs. It is also obvious that Fig. 4 supports hybrid ANNs–ES systems (along the ANN–ES borderline).

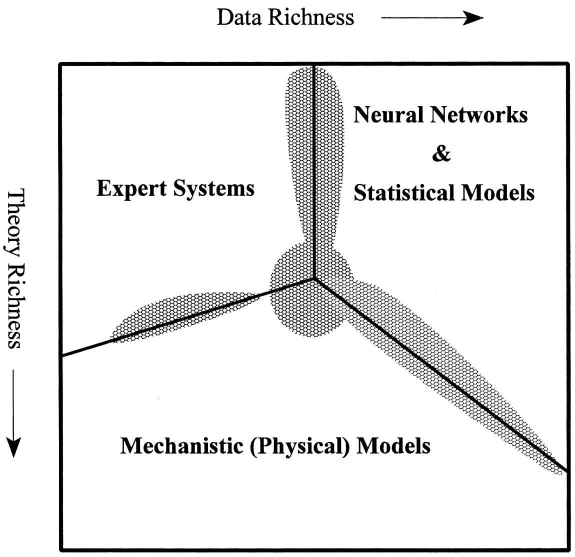


Fig. 4. Suitability of modeling technique in relation to data and theory richness (adapted from Rumelhart et al., 1995).

5. Challenging problems

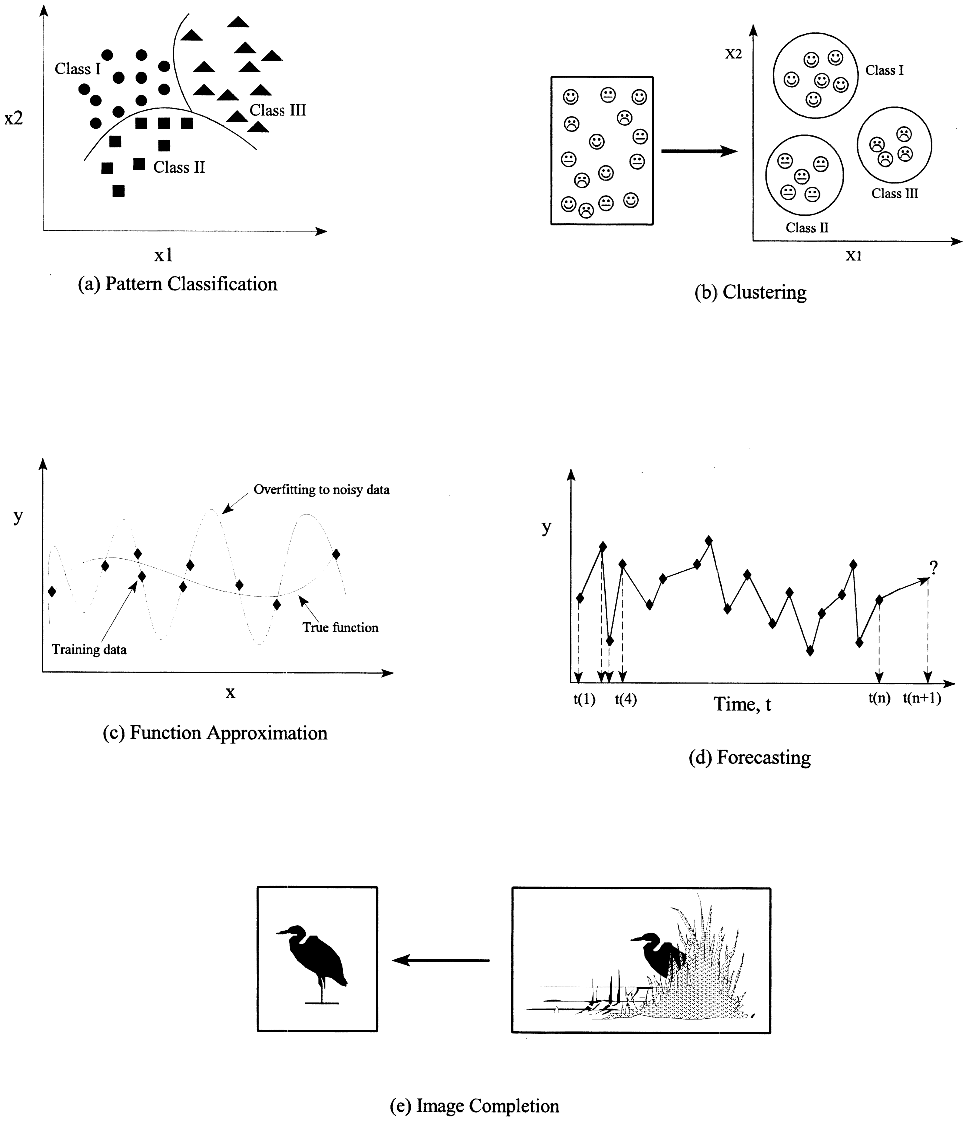
Generally, ANNs are more robust and often outperform other computational tools in solving a variety of problems from seven categories.

5.1. Pattern classification

Pattern classification deals with assigning an unknown input pattern, using supervised learning, to one of several prespecified classes based on one or more properties that characterize a given class, as shown in Fig. 5a. Classification applications from the area of microbiology include classification of commodities based on their microbiological characteristics (e.g., Jonsson et al., 1997), and characterization of microorganisms using pyrolysis mass spectrometry data (e.g., Chun et al., 1993a,b). Unlike discriminant analysis in statistics, ANNs do not require the linearity assumption and can be applied to nonlinearly separable classes (Garth et al., 1996).

5.2. Clustering

Clustering is performed via unsupervised learning in which the clusters (classes) are formed by exploring the similarities or dissimilarities between the input patterns based on their inter-correlations (Fig. 5b). The network assigns 'similar' patterns to the same cluster. Example applications from microbiology include sub-species discrimination using



(a) Pattern Classification

(b) Clustering

(c) Function Approximation

(d) Forecasting

(e) Image Completion

Fig. 5. Problems solved by ANNs. (a) Pattern classification. (b) Clustering. (c) Function approximation. (d) Forecasting. (e) Association (e.g., image completion).

pyrolysis mass spectrometry and Kohonen networks (Goodacre et al., 1994).

5.3. Function approximation (modeling)

Function approximation (modeling) involves training ANN on input–output data so as to approximate the underlying rules relating the inputs to the outputs (Fig. 5c). Multilayer ANNs are considered universal approximators that can approximate any arbitrary function to any degree of accuracy (Hecht-Nielsen, 1990), and thus are normally used in this application. Function approximation is applied to problems (i) where no theoretical model is available, i.e., data obtained from experiments or observations are utilized, or (ii) to substitute theoretical models that are hard to compute analytically by utilizing data obtained from such models. Examples from this category are numerous in microbiology, e.g., predicting microbial growth (Geeraerd et al., 1998; Hajmeer et al., 1996, 1997, 1998).

5.4. Forecasting

Forecasting includes training of an ANN on samples from a time series representing a certain phenomenon at a given scenario and then using it for other scenarios to predict (forecast) the behavior at subsequent times (Fig. 5d). That is, the network will predict $Y(t+1)$ from one or more previously known historical observations [e.g., $Y(t-2)$, $Y(t-1)$, and $Y(t)$, where t is the time step]. Microbial growth curves can be modeled in such a manner (Hajmeer et al., 2000).

5.5. Optimization

Optimization is concerned with finding a solution that maximizes or minimizes an objective function subject to a set of constraints. Optimization is a well-established field in mathematics, however ANNs, such as the Hopfield network (Hopfield and Tank, 1986), were found to be more efficient in solving complex and nonlinear optimization problems (Pham, 1994).

5.6. Association

Association involves developing a pattern associator ANN by training on ideal noise-free data and subsequently using this ANN to classify noise-corrupted data (e.g., for novelty detection). The associative network may also be used to correct (reconstruct) the corrupted data or completely missing data (or image), as shown in Fig. 5e. Hopfield and Hamming networks are especially used for this application (Lippmann, 1987), and to a lesser degree multilayer backpropagation ANNs trained on patterns with identical input and output (Fu, 1995).

5.7. Control

Control is concerned with designing a network, normally recurrent, that will aid an adaptive control system to generate the required control inputs such that the system will follow a certain trajectory based on system feedback (Jain et al., 1996).

6. Classification of ANNs

ANNs may be classified in many different ways according to one or more of their relevant features. Generally, classification of ANNs may be based on (i) the function that the ANN is designed to serve (e.g., pattern association, clustering), (ii) the degree (partial/full) of connectivity of the neurons in the network, (iii) the direction of flow of information within the network (recurrent and nonrecurrent), with recurrent networks being dynamic systems in which the state at any given time is dependent on previous states, (iv) the type of learning algorithm, which represents a set of systematic equations that utilize the outputs obtained from the network along with an arbitrary performance measure to update the internal structure of the ANN, (v) the learning rule (the driving engine of the learning algorithm), and (vi) the degree of learning supervision needed for ANN training. Supervised learning involves training of an ANN with the correct answers (i.e., target outputs) being given for every example, and using the deviation (error) of the ANN solution from corresponding target values to determine the required amount by which each weight should be adjusted. Reinforcement learning is supervised, however the ANN is provided with a critique on correctness of output rather than the correct answer itself. Unsupervised learning does not require a correct answer for the training examples, however the network, through exploring the underlying structure in the data and the correlation between the various examples, organizes the examples into clusters (categories) based on their similarity or dissimilarity (e.g., Kohonen networks). Finally, the hybrid learning procedure combines supervised and unsupervised learning.

As examples of classification, Lippmann (1987) classified ANNs with regard to learning (supervised vs. unsupervised) and data (binary vs. continuous). Simpson (1990) categorized ANNs with respect to learning (supervision) and the flow of data in the network (feedforward vs. feedback). Maren (1991) proposed a hierarchal categorization based on structure followed by dynamics, then learning. Jain et al.

(1996) present a four-level classification based on the degree of learning supervision, the learning rule, data flow in the ANN, and the learning algorithm.

7. Learning rules

A learning rule defines how exactly the network weights should be adjusted (updated) between successive training cycles (epochs). There are four basic types of rules (Hassoun, 1995; Haykin, 1994). The error-correction learning (ECL) rule is used in supervised learning in which the arithmetic difference (error) between the ANN solution at any stage (cycle) during training and the corresponding correct answer is used to modify the connection weights so as to gradually reduce the overall network error. The Boltzmann learning (BL) rule is a stochastic rule derived from thermodynamic principles and information theory (Anderson and Rosenfeld, 1988). It is similar to ECL, however each neuron generates an output (or state) based on a Boltzmann statistical distribution (Jain et al., 1996), which renders learning extremely slower. The Hebbian learning (HL) rule (Hebb, 1949), developed based on neurobiological experiments, is the oldest learning rule, which postulates that ‘‘if neurons on both sides of a synapse are activated synchronously and repeatedly, the synapse’s strength is selectively increased.’’ Therefore, unlike ECL and BL rules, learning is done locally by adjusting the synapse weight based on the activities of the neurons. In the competitive learning (CL) rule, all neurons are forced to compete among themselves such that only one neuron will be activated in a given iteration with all the weights attached to it adjusted (Jain et al., 1996). The CL rule is speculated to exist in many biological systems (Haykin, 1994).

8. Popular ANNs

A vast number of networks, new or modifications of existing ones, are being constantly developed. Simpson (1990) listed 26 different types of ANNs, and Maren (1991) listed 48. Pham (1994) estimated that over 50 different ANN types exist. Some applications may be solved using different ANN types, whereas others may only be solved via a specific ANN type. Some networks are more proficient in solving perceptual problems, while others are more suitable for data modeling and function approximation. A brief discussion of the most frequently used ANNs, presented in the order of their discovery, is given below.

8.1. Hopfield networks

This network is a symmetric fully connected two-layer recurrent network that acts as a nonlinear associative memory and is especially efficient in solving optimization problems (Hopfield, 1984; Hopfield and Tank, 1986). The network is suited to only bipolar or binary inputs and it implements an energy function. Learning is done by setting each weight connecting two neurons to the product of the inputs of these two neurons (van Rooij et al., 1996). When presented with an incomplete or noisy pattern, the network responds by retrieving an internally stored pattern that most closely resembles the presented pattern.

8.2. Adaptive resonance theory (ART) networks

These are trained by unsupervised learning where the network adapts to the information environment without intervention. The ART network consists of two fully interconnected layers, a layer that receives the inputs and a layer consisting of output neurons. The feedforward weights are used to select the winning output neuron (cluster) and serve as the long-term memory for the networks. The feedback weights are the vigilance weights that are used to test the vigilance and serve as the short-term memory for the network (van Rooij et al., 1996). An ART network stores a set of patterns in such a way that when the network is presented with a new pattern it will either match it to a previously stored pattern, or store it as a new pattern if it is sufficiently dissimilar to the closest pattern (Carpenter and Grossberg, 1987, 1988). Like Hopfield nets, ART networks can be used for pattern recognition, completion, and classification.

8.3. Kohonen networks

These networks, also called self-organizing feature maps, are two-layer networks that transform *n*-dimensional input patterns into lower-ordered data where similar patterns project onto points in close proximity to one another (Kohonen, 1989). Kohonen networks are trained in an unsupervised manner to form clusters within the data (i.e., data grouping). In addition to pattern recognition and classification, Kohonen maps are used for data compression, in which high-dimensional data are mapped into a fewer dimensions space while preserving their content (Zupan and Gasteiger, 1991).

8.4. Backpropagation networks

These networks are the most widely used type of networks and are considered the workhorse of ANNs (Rumelhart et al., 1986). A backpropagation (BP) network is an MLP consisting of (i) an input layer with nodes representing input variables to the problem, (ii) an output layer with nodes representing the dependent variables (i.e., what is being modeled), and (iii) one or more hidden layers containing nodes to help capture the nonlinearity in the data. Using supervised learning (with the ECL rule), these networks can learn the mapping from one data space to another using examples. The term backpropagation refers to the way the error computed at the output side is propagated backward from the output layer, to the hidden layer, and finally to the input layer. In BPANNs, the data are fed forward into the network without feedback (i.e., all links are unidirectional and there are no same layer neuron-to-neuron connections). The neurons in BPANNs can be fully or partially interconnected. These networks are so versatile and can be used for data modeling, classification, forecasting, control, data and image compression, and pattern recognition (Hassoun, 1995).

8.5. Recurrent networks

In a recurrent network, the outputs of some neurons are fed back to the same neurons or to neurons in preceding layers. This enables a flow of information in both forward and backward directions, thus providing the ANN with a dynamic memory (Pham, 1994). There are special algorithms for training recurrent networks (Hassoun, 1995; Hecht-Nielsen, 1990). The BP recurrent ANNs are a simple variant of recurrent networks in which the 'memory' is introduced into static feedforward ANNs by a special data representation (e.g., time delay) followed by training using classic BP (Basheer, 2000; ASCE, 2000).

8.6. Counterpropagation networks

These networks, developed by Hecht-Nielsen (1988, 1990), are trained by hybrid learning to create a self-organizing look-up table useful for function approximation and classification (Zupan and Gasteiger, 1993). As input features are presented to the network, unsupervised learning is carried out to create a Kohonen map of the input data. Meanwhile, supervised learning is used to associate an appropriate output vector with each point on the map. Once the network has been trained, each newly presented feature vector will trigger a response which is the average for those feature vectors closest to it in the input data space, thus simulating a look-up table.

8.7. Radial basis function (RBF) networks

These networks are a special case of a multilayer feedforward error-backpropagation network with three layers (Schalkoff, 1997). They can be trained by a variety of learning algorithms including a two-step hybrid learning (Haykin, 1994). The hidden layer is used to cluster the inputs of the network (the nodes in this layer are called cluster centers). Unlike the sigmoid transfer function in BPANNs, these networks employ a radial basis function such as a Gaussian kernel (Haykin, 1994). The RBF is centered at the point specified by the weight vector associated with the unit. Both the positions and widths of these Gaussian functions must be learnt from the training patterns. Each output unit implements a linear combination of these RBFs. The choice between the RBF networks and the BPANNs is problem dependent (Pal and Srimani, 1996). RBF networks train faster than BP but are not as versatile and are comparatively slower for use (Attoh-Okine et al., 1999).

The decision as to which network works better for a given problem depends strictly on the problem logistics. For example, a clustering problem requires a Kohonen network, a mapping problem may be modeled using a variety of ANNs such as BP and RBF networks, and some optimization problems may only be solved using Hopfield networks. Other factors governing ANN selection are the input type (i.e., whether it is boolean, continuous, or a mixture), and the execution speed of the network once trained and implemented in serial hardware. Other issues for ANN selection are discussed by Hudson and Postma (1995).

9. Backpropagation ANNs

To extend the understanding of ANNs from the level of identifying what these systems are to how to design them, it is imperative to become familiar with ANN computation and design. For this objective, the BPANNs are discussed in more detail, for their popularity, and their flexibility and adaptability in modeling a wide spectrum of problems in many application areas.

The feedforward error-backpropagation learning algorithm is the most famous procedure for training ANNs. BP is based on searching an error surface (error as a function of ANN weights) using gradient descent for point(s) with minimum error. Each iteration in BP constitutes two sweeps: forward activation to produce a solution, and a backward propagation of the computed error to modify the weights. In an initialized ANN (i.e., an ANN with assumed initial weights), the forward sweep involves presenting the network with one training example. This starts at the input layer where each input node transmits the value received forward to each hidden node in the hidden layer. The collective effect on each of the hidden nodes is summed up by performing the dot product of all values of input nodes and their corresponding interconnection weights, as described in Eq. (1). Once the net effect at one hidden node is determined, the activation at that node is calculated using a transfer function (e.g., sigmoidal function) to yield an output between 0 and $+1$ or -1 and $+1$. The amount of activation obtained represents the new signal that is to be transferred forward to the subsequent layer (e.g., either hidden or output layer). The same procedure of calculating the net effect is repeated for each hidden node and for all hidden layers. The net effect(s) calculated at the output node(s) is consequently transformed into activation(s) using a transfer function. The activation(s) just calculated at the output node(s) represents the ANN solution of the fed example, which may deviate considerably from the target solution due to the arbitrary selected interconnection weights. In the backward sweep, the difference (i.e., error) between the ANN and target outputs is used to adjust the interconnection weights, starting from the output layer, through all hidden layers, to the input layer, as will be described in the following section. The forward and backward sweeps are performed repeatedly until the ANN solution agrees with the target value within a prespecified tolerance. The BP learning algorithm provides the needed weight adjustments in the backward sweep.

9.1. BP Algorithm

Because of its importance and simplicity, the BP algorithm will be presented here in its final form. Complete derivation of the algorithm can be found elsewhere (e.g., Zupan and Gasteiger, 1993; Haykin, 1994) and a clear systematic derivation is given by Basheer (1998) and Najjar et al. (1997). In order to be able to run the algorithm, it is essential to define the interlayer as the gap between two successive layers that encloses the connection weights and contains only the neurons of the upper layer, as shown in Fig. 6 (assuming that all layers are positioned above the input layer). Consider an MLP network with L interlayers. For interlayer $l \in \{1,2,\ldots,L\}$ there are N_l nodes and $N_l \times N_{l-1}$ connection links with weights $\boldsymbol{W} \in R^{N_l \times N_{l-1}}$, where N_l and N_{l-1} are the numbers of nodes (including thresholds) in interlayers l and $l-1$, respectively (Fig. 6). A connection weight is denoted by W_{ji}^{l} if it resides in interlayer l and connects node j of interlayer l with node i of lower (preceding) interlayer $l-1$ (node i is the source node and node j is the destination node). In any interlayer l, a typical neuron j integrates the signals, x_j, impinging onto it, and produces a net effect, ξ_j, according to linear neuron dynamics:

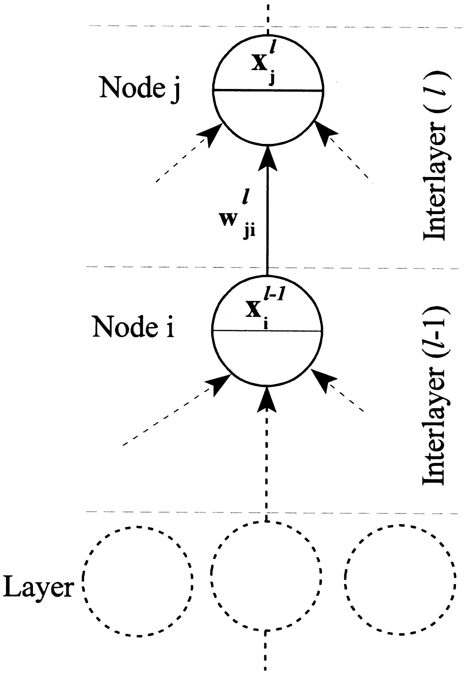


Fig. 6. Notation and index labeling used in backpropagation ANNs.

$$\xi_j^l = \sum_{i=1}^{N_{l-1}} w_{ji}^l x_i^{l-1}. \tag{2}$$

The corresponding activation, x_j^l, of the neuron is determined using a transfer function, σ, that converts the total signal into a real number from a bounded interval:

$$x_j^l = \sigma(\xi_j^l) = \sigma\left(\sum_{i=1}^{N_{l-1}} w_{ji}^l x_i^{l-1}\right). \tag{3}$$

One popular function used in BP is the basic continuous sigmoid:

$$\sigma(\xi) = \frac{1}{1+e^{-\xi}}, \tag{4}$$

where $-\infty < \xi < \infty$ and $0.0 \leq \sigma \leq 1.0$. Eqs. (2)–(4) are used for all nodes to calculate the activation. For the input nodes the activation is simply the raw input. In any interlayer, an arbitrary weight w_{ji}^l at iteration (t) will be updated from its previous state ($t-1$) value according to

$$w_{ji}^l(t) = w_{ji}^l(t-1) + \Delta w_{ji}^l(t), \tag{5}$$

where Δw_{ji}^l is the $(+/-)$ incremental change in the weight. The weight change is determined via the modified delta rule (Zupan and Gasteiger, 1993), which can be written as

$$\Delta w_{ji}^l = \eta \delta_j^l x_i^{l-1} + \mu \Delta w_{ji}^{l(\text{previous})}, \tag{6}$$

where η is the learning rate controlling the update step size, μ is the momentum coefficient, and x_i^{l-1} is the input from the $l-1$th interlayer. The first part of the right-hand side of Eq. (6) is the original delta rule. The added momentum term helps direct the search on the error hyperspace to the global minimum by allowing a portion of the previous updating (magnitude and direction) to be added to the current updating step. Note that Eq. (6) can also be applied to any neuron threshold (bias) which can be assumed as a link, with weight equal to the threshold value, for an imaginary neuron whose activation is fixed at 1.0. The weight change can also be determined using a gradient descent written in generalized form for an interlayer l:

$$\Delta w_{ji}^l = -\kappa\left(\frac{\partial \varepsilon^l}{\partial w_{ji}^l}\right). \tag{7}$$

Therefore, in order to determine the incremental changes for the lth interlayer, the main task is to quantify the error gradient $(\partial \varepsilon^l / \partial w_{ji}^l)$. Using Eqs. (6) and (7), the required weight change can be derived with different expressions depending on whether the considered neuron is in the output layer or in a hidden layer. If the neuron is in the output layer, then $l = L$ in Eq. (6), with δ_j^L calculated from

$$\delta_j^L = (x_j^L - y_j) x_j^L (1 - x_j^L). \tag{8}$$

If the neuron is in a hidden layer, the weight change is also calculated using Eq. (6) with δ_j^l determined from

$$\delta_j^l = x_j^l (1 - x_j^l)\left(\sum_{k=1}^{r} \delta_k^{l+1} w_{kj}^{l+1}\right), \tag{9}$$

where δ_k^{l+1} is calculated for a given non-output layer (l) beginning with a layer one level up ($l+1$) and moving down layer by layer. That is, for the last (uppermost) hidden layer in a network, δ_j^l is determined via δ_k^{l+1} of the output layer calculated using Eq. (8). The above delta equations (Eqs. (8) and (9)) are based on the sigmoid transfer function given in Eq. (4). For a different function, the terms $x_j^L(1-x_j^L)$ and $x_j^l(1-x_j^l)$ in Eqs. (8) and (9), respectively, should be replaced with the relevant first derivative of the used function. This technique of distributing backward the errors starting from the output layer down through the hidden layer gave the method the name backpropagation of error with the modified delta rule (Rumelhart et al., 1986). The standard BP have been modified in several ways to achieve a better search and accelerate and stabilize the training process (Looney, 1996; Masters, 1994).

10. ANN development project

The development of a successful ANN project constitutes a cycle of six phases, as illustrated in Fig. 7. The problem definition and formulation (phase 1) relies heavily on an adequate understanding of the problem, particularly the 'cause–effect' relationships. The benefits of ANNs over other techniques (if available) should be evaluated before final selection of the modeling technique. System design (phase 2) is the first step in the actual ANN design in which the modeler determines the type of ANN and learning rule that fit the problem. This phase also involves data collection, data preprocessing to fit the type of ANN used, statistical analysis of data, and partitioning the data into three distinct subsets (training, test, and validation subsets). System realization (phase 3) involves training of the network utilizing the training and test subsets, and simultaneously assessing the network performance by analyzing the prediction error. Optimal selection of the various parameters (e.g., network size, learning rate, number of training cycles, acceptable error, etc.) can affect the design and performance of the final network. Splitting the problem into smaller sub-problems, if possible, and designing an ensemble of networks could enhance the overall system accuracy. This takes the modeler back to phase 2. In system verification (phase 4), although network development includes ANN testing against the test data while training is in progress, it is good practice (if data permit) that the 'best' network be examined for its generalization capability using the validation subset. Verification is intended to confirm the capability of the ANN-based model to respond accurately to examples never used in network development. This phase also includes comparing the performance

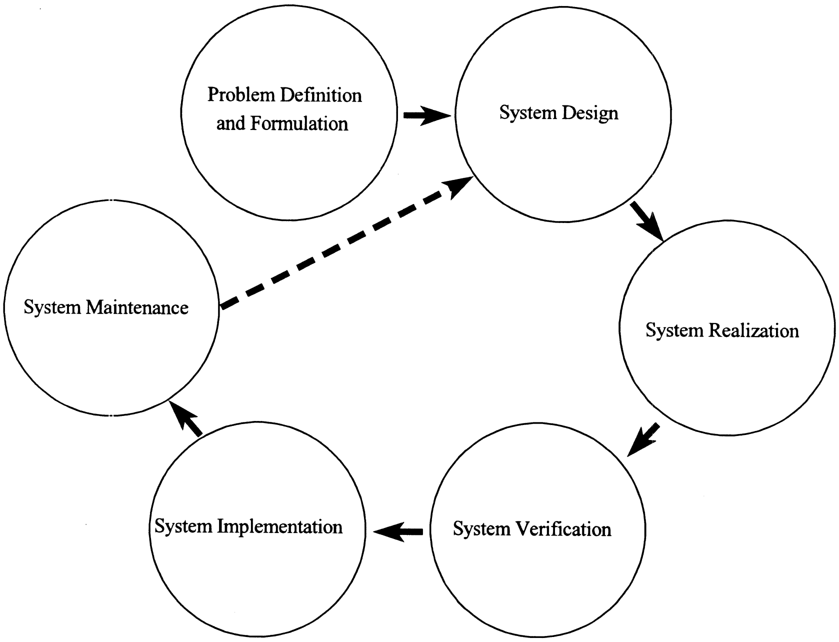


Fig. 7. The various phases in an ANN development project.

of the ANN-based model to those of other approaches (if available) such as statistical regression and expert systems. System implementation (phase 5) includes embedding the obtained network in an appropriate working system such as hardware controller or computer program. Final testing of the integrated system should also be carried out before its release to the end user. System maintenance (phase 6) involves updating the developed system as changes in the environment or the system variables occur (e.g., new data), which involves a new development cycle.

11. General issues in ANN development

A number of issues should be addressed before initiation of any network training. Some of the following issues are only relevant to BP ANNs while others are applicable to the design of all ANN types.

11.1. Database size and partitioning

Models developed from data generally depend on database size. ANNs, like other empirical models, may be obtained from databases of any size, however generalization of these models to data from outside the model development domain will be adversely affected. Since ANNs are required to generalize for unseen cases, they must be used as interpolators. Data to be used for training should be sufficiently large to cover the possible known variation in the problem domain.

The development of an ANN requires partitioning of the parent database into three subsets: training, test, and validation. The training subset should include all the data belonging to the problem domain and is used in the training phase to update the weights of the network. The test subset is used during the learning process to check the network response for untrained data. The data used in the test subset should be distinct from those used in the training, however they should lie within the training data boundaries. Based on the performance of the ANN on the test subset, the architecture may be changed and/or more training cycles applied. The third portion of the data is the validation subset which should include examples different from those in the other two subsets. This subset is used after selecting the best network to further examine the network or confirm its accuracy before being implemented in the neural system and/or delivered to the end user.

Currently, there are no mathematical rules for the determination of the required sizes of the various data subsets. Only some rules of thumb derived from experience and analogy between ANNs and statistical regression exist. Baum and Haussler (1989) propose the minimum size of the training subset to be equal to the number of weights in the network times the inverse of the minimum target error. Dowla and Rogers (1995) and Haykin (1994) suggest an example-to-weight ratio (EWR) >10, while Masters (1994) suggests EWR >4. For database partitioning, a large test subset may highlight the generalization capability better; however, the remaining smaller training subset may not be adequate to train the network satisfactorily. Looney (1996) recommends 65% of the parent database to be used for training, 25% for testing, and 10% for validation, whereas Swingler (1996) proposes 20% for testing and Nelson and Illingworth (1990) suggest 20–30%.

11.2. Data preprocessing, balancing, and enrichment

Several preprocessing techniques are usually applied before the data can be used for training to accelerate convergence. Among these are noise removal, reducing input dimensionality, and data transformation (Dowla and Rogers, 1995; Swingler, 1996), treatment of non-normally distributed data, data inspection, and deletion of outliers (Masters, 1994; Stein, 1993). Balancing of data is especially important in classification problems. It is desired that the training data be distributed nearly evenly between the various classes to prevent the network from being biased to the over-represented classes (Swingler, 1996). To balance a database, some of the over-represented classes may be removed or extra examples pertaining to the under-represented class added. Another way is by duplicating the under-represented input/output examples and adding random noise to their input data (while keeping the output class unchanged). Swingler (1996) suggests the use of information theory to measure the degree of balance of the training database.

Small database size poses another problem in ANN development because of the inability to partition the database into fairly-sized subsets for training, test, and validation. To expand the size of the database, the trivial way is to get new data (if possible) or interject random noise in the available examples to generate new ones. Noise addition normally enhances the ANN robustness against measurement error (e.g., noise = ± instrument sensitivity). If data enrichment is not possible, the leave-one-out method (or leave-*k*-out method) may be used for developing a network (Hecht-Nielsen, 1990; Rizzo and Dougherty, 1994). With M exemplars available, a network is trained on $M-1$ (or $M-k$) exemplars, and tested on the one (or k) unused exemplar(s). The procedure is repeated M times, each with a new set of randomly initialized weights. The solutions of the M networks are then averaged to obtain a representative solution to the problem. Other techniques to train and validate networks with limited data include grouped cross-validation, grouped jackknife, and bootstrap (Twomey and Smith, 1997).

11.3. Data normalization

Normalization (scaling) of data within a uniform range (e.g., 0–1) is essential (i) to prevent larger numbers from overriding smaller ones, and (ii) to prevent premature saturation of hidden nodes, which impedes the learning process. This is especially true when actual input data take large values. There is no one standard procedure for normalizing inputs and outputs. One way is to scale input and output variables (z_i) in interval $[\lambda_1, \lambda_2]$ corresponding to the range of the transfer function:

$$x_i = \lambda_1 + (\lambda_2 - \lambda_1)\left(\frac{z_i - z_i^{\min}}{z_i^{\max} - z_i^{\min}}\right), \tag{10}$$

where x_i is the normalized value of z_i, and $z_i^{\max}$ and $z_i^{\min}$ are the maximum and minimum values of z_i in the database. It is recommended that the data be normalized between slightly offset values such as 0.1 and 0.9 rather than between 0 and 1 to avoid saturation of the sigmoid function leading to slow or no learning (Hassoun, 1995; Masters, 1994). Other more computationally involved techniques are given by Masters (1994), Swingler (1996), and Dowla and Rogers (1995). Masters (1994) indicates that more complicated techniques may not produce any better solution than that obtained using linear normalization (Eq. (10)). For parameters with an exceptionally large range, it may be beneficial to take the logarithm of data prior to normalization [if data contain zeros, $\log(z_i + 1)$ may be used].

11.4. Input/output representation

Proper data representation also plays a role in the design of a successful ANN (Masters, 1994). The data inputs and outputs can be continuous, discrete, or a mixture of both. For example, in a classification problem where each of the input variables belongs to one of several classes and the output is also a class, all the inputs and outputs may be represented by binary numbers such as 0 and 1 (or 0.1 and 0.9 to prevent saturation). If two inputs (A and B) are to be assigned to four levels of activation (e.g., low, medium, high, and very high), then each input may be represented by two binary numbers such as 00, 01, 10, and 11 to indicate the four levels. Another representation may assign four binary numbers to each input such as 0001, 0010, 0100, and 1000 where the location of 1 determines the type of activation of the input variable. Similar treatment applies to the output variables. This representation increases the dimensionality of the input vector (the two-digit representation converts the input vector into four inputs, and the four-digit representation into eight inputs). Binary inputs and outputs are very useful in extracting rules from a trained network (Fu, 1995). For this purpose, a continuous variable may be replaced by binary numbers by partitioning its range into a number of intervals, each assigned to a unique class. Specialized algorithms for discretizing variables based on their distribution also exist (Kerber, 1992).

11.5. Network weight initialization

Initialization of a network involves assigning initial values for the weights (and thresholds) of all connections links. Some researchers (e.g., Li et al., 1993; Schmidt et al., 1993) indicate that weights initialization can have an effect on network convergence. Hassoun (1995) explained that if the initial weight vector is stationed in a flat region of the error surface the convergence may become extremely slow. Other studies (e.g., Fahlman, 1988) have shown that initialization has an insignificant effect on both the convergence and final network architecture. Typically, weights and thresholds are initialized uniformly in a relatively small range with zero-mean random numbers (Rumelhart et al., 1986). However, an extremely small range can lead to very small error gradients which may slow down the initial learning process. The choice of small numbers is very essential to reduce the likelihood of premature neurons saturation (Lee et al., 1991). ASCE (2000) recommends that weights and thresholds be assigned initial small random values between -0.30 and $+0.30$. Weight initialization can also be performed on a neuron-by-neuron basis (Haykin, 1994) by assigning values uniformly sampled from the range $(-r/N_j, +r/N_j)$, where r is a real number depending on the neuron activation function, and N_j is the number of connections feeding into neuron j. Wessels and Barnard (1992) use zero-mean and unit standard deviation for links feeding neurons with weights sampled from $[-3M^{-1/2}, +3M^{-1/2}]$, where M is the number of weights in a given interlayer. Nguyen and Widrow (1990) initialize the weight vector so that each input exemplar is likely to force a hidden unit to learn efficiently.

11.6. BP learning rate (η)

A high learning rate, η, will accelerate training (because of the large step) by changing the weight vector, W, significantly from one cycle to another. However, this may cause the search to oscillate on the error surface and never converge, thus increasing the risk of overshooting a near-optimal W. In contrast, a small learning rate drives the search steadily in the direction of the global minimum, though slowly. A constant learning rate may be utilized throughout the training process. Wythoff (1993) suggests $\eta = 0.1$–10, Zupan and Gasteiger (1993) recommend $\eta = 0.3$–0.6, and Fu (1995) recommends $\eta = 0.0$–1.0. The adaptive learning rate $[\eta(t)]$, which varies along the course of training, could also be used and can be effective in achieving an optimal weight vector for some problems. Generally, larger steps are needed when the search is far away from a minimum, and smaller steps as the search approaches a minimum. Since the distance from a minimum cannot be predicted, various heuristics have been proposed (Hassoun, 1995; Haykin, 1994).

11.7. BP momentum coefficient (μ)

A momentum term is commonly used in weight updating to help the search escape local minima and reduce the likelihood of search instability (Haykin, 1994; Zupan and Gasteiger, 1993). As implied in Eq. (6), μ accelerates the weight updates when there is a need to reduce η to avoid oscillation. A high μ will reduce the risk of the network being stuck in local minima, but it increases the risk of overshooting the solution as does a high learning rate. A $\mu > 1.0$ yields excessive contributions of the weight increments of the previous step and may cause instability (Henseler, 1995). Conversely, an extremely small μ leads to slow training. Both a constant and adaptable momentum can be utilized. Wythoff (1993) suggests $\mu = 0.4$–0.9, Hassoun (1995) and Fu (1995) suggest $\mu = 0.0$–1.0, Henseler (1995) and Hertz et al. (1991) suggest $\mu \approx 1.0$, and Zupan and Gasteiger (1993) suggest that $\eta + \mu \approx 1$. Swingler (1996) uses $\mu = 0.9$ and $\eta = 0.25$ in solving all problems unless a good solution could not be obtained. Depending on the problem being solved, it seems that the success of training varies with the selected μ, and a trial-and-error procedure is normally preferred. Adaptive momentum involves varying μ with the training cycle [i.e., $\mu(t)$] in which the changes in μ are made in relation to error gradient information (Fahlman, 1988; Yu et al., 1993). Other methods relate μ to the adaptive learning rate η such that μ is decreased when learning speeds up (Eaton and Olivier, 1992). Finally, the addition of momentum should be considered with caution because of the need of doubling computer space for storing weights of current and previous iterations (see Eq. (6)).

11.8. Transfer function, σ

The transfer (activation) function, σ, is necessary to transform the weighted sum of all signals impinging onto a neuron so as to determine its firing intensity. Some functions are designed to indicate only whether a neuron can fire (step functions) regardless of the magnitude of the net excitation (ξ) by comparing ξ to the neuron threshold (Eq. (1)). Most applications utilizing BPANNs employ a sigmoid function, which possesses the distinctive properties of continuity and differentiability on $(-\infty,\infty)$, essential requirements in BP learning. Moody and Yarvin (1992) reported various success rates with different transfer functions in relation to data nonlinearity and noisiness. Han et al. (1996) use a variant logistic function with three adjustable parameters, and each neuron is assigned a different set of values for these parameters. The advantage of choosing a particular transfer function over another is not yet theoretically understood (Hassoun, 1995).

11.9. Convergence criteria

Three different criteria may be used to stop training: (i) training error ($\rho \le \varepsilon$), (ii) gradient of error ($\nabla\rho \le \delta$), and (iii) cross-validation, where ρ is the arbitrary error function, and ε and δ are small real numbers. The third criterion is more reliable, however it is computationally more demanding and often requires abundant data. Convergence is usually based on the error function, ρ, exhibiting deviation of the predictions from the corresponding target output values such as the sum of squares of deviations. Training proceeds until ρ reduces to a desired minimum. The function ρ may also be expressed as the relative error of the absolute values of the deviations averaged over the subset. Another criterion is the coefficient-of-determination, R^2, representing the agreement between the predicted and target outputs. Other more involved methods for monitoring network training and generalization are based on information theory (Swingler, 1996).

The most commonly used stopping criterion in neural network training is the sum-of-squared-errors (SSE) calculated for the training or test subsets as

$$\mathrm{SSE} = \frac{1}{N}\sum_{p=1}^{N}\sum_{i=1}^{M}(t_{pi} - O_{pi})^2, \tag{11}$$

where O_{pi} and t_{pi} are, respectively, the actual and target solution of the ith output node on the pth example, N is the number of training examples, and M is the number of output nodes. Some SSE criteria incorporate a measure of complexity of the network architecture (Garth et al., 1996). Generally, the error on training data decreases indefinitely with increasing number of hidden nodes or training cycles, as shown in Fig. 8. The initial large drop in error is due to learning, but the subsequent slow reduction in error may be attributed to (i) network memorization resulting from the excessively large number of training cycles used, and/or (ii) overfitting due to the use of a large number of hidden nodes. During ANN training, the error on test subsets is monitored which generally shows an initial reduction and a subsequent increase due to memorization and overtraining of the trained ANN. The final (optimal) neural network architecture is obtained at the onset of the increase in test data error.

Other error metrics may be used and may perform equally well in terms of optimizing network structure (Twomey and Smith, 1997). For classification problems (discrete-valued output), the convergence criterion should be based on the hit (or miss) rate representing the percentage of examples classified correctly (or incorrectly), or confusion matrices (Lakshmanan, 1997), rather than the absolute deviation of the network classification from the target classification.

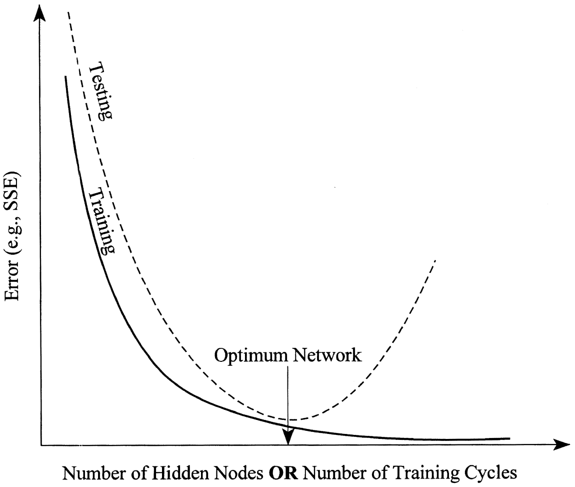


Fig. 8. Criteria for termination of training and selection of optimum network architecture.

11.10. Number of training cycles

The number of training cycles required for proper generalization may be determined by trial and error. For a given ANN architecture, the error in both training and test data is monitored for each training cycle. Training for so long can result in a network that can only serve as a look-up table, a phenomenon called overtraining or memorization (Zupan and Gasteiger, 1993; Wythoff, 1993). Theoretically, excessive training can result in near-zero error on predicting the training data (called recall), however generalization on test data may degrade significantly (Fig. 8). Initially, the test subset error continues to decrease with the number of training cycles. As the network loses its ability to generalize on the test data, the error starts to build up after each epoch. Although the error on the test data may not follow a smooth path, the onset of a major increase in the error is considered to represent the optimal number of cycles for that ANN architecture.

11.11. Training modes

Training examples are presented to the network in either one or a combination of two modes: (i) example-by-example training (EET), and (ii) batch training (BT) (Zupan and Gasteiger, 1993; Wythoff, 1993). In EET mode, the weights are updated immediately after the presentation of each training example. Here, the first example is presented to the network, and the BP learning algorithm consisting of feedforward and backward sweeps is applied for either a specified number of iterations or until the error drops to the desired level. Once the first example is learnt, the second example is presented and the procedure is repeated. The advantages of EET include the smaller storage requirements for the weights as opposed to BT, and the better stochastic search, which prevents entrapment in local minima (Zupan and Gasteiger, 1993). The disadvantage of EET is associated with the fact that learning may become stuck in a first very bad example, which may force the search in the wrong direction. Conversely, BT requires that weight updating be performed after all training examples have been presented to the network. That is, the first learning cycle will include the presentation of all the training examples, the error is averaged over all the training examples (e.g., Eq. (11)), and then backpropagated according to the BP learning law. Once done, the second cycle includes another presentation of all examples, and so on. The advantages of the BT mode include a better estimate of the error gradient vector and a more representative measurement of the required weight change. However, this training mode requires a large storage of weights, and is more likely to be trapped in a local minimum (Zupan and Gasteiger, 1993). For a better search, the order of presentation of the training examples may be randomized between successive training cycles (Zupan and Gasteiger, 1991). The effectiveness of the two training modes can be problem specific (Hertz et al., 1991; Haykin, 1994; Swingler, 1996).

11.12. Hidden layer size

In most function approximation problems, one hidden layer is sufficient to approximate continuous functions (Basheer, 2000; Hecht-Nielsen, 1990). Generally, two hidden layers may be necessary for learning functions with discontinuities (Masters, 1994). The determination of the appropriate number of hidden layers and number of hidden nodes (NHN) in each layer is one of the most critical tasks in ANN design. Unlike the input and output layers, one starts with no prior knowledge as to the number and size of hidden layers. As shown in Fig. 9, a network with too few hidden nodes would be incapable of differentiating between complex patterns leading to only a linear estimate of the actual trend. In contrast, if the network has too many hidden nodes it will follow the noise in the data due to overparameterization leading to poor generalization for untrained data (Fig. 9). With increasing number of hidden nodes, training becomes excessively time-consuming. The optimal number of HN essential for network generalization may be a function of input/output vector sizes, size of training and test subsets, and, more importantly, the problem of nonlinearity. Several rules of thumb are available in the literature which relate hidden layer size to the number of nodes in input (N_{INP}) and output (N_{OUT}) layers. Jadid and Fairbairn (1996) called for an upper bound on NHN equal to $N_{\mathrm{TRN}}/[R+(N_{\mathrm{INP}}+N_{\mathrm{OUT}})]$, where N_{TRN} is the number of training patterns and $R=5$–

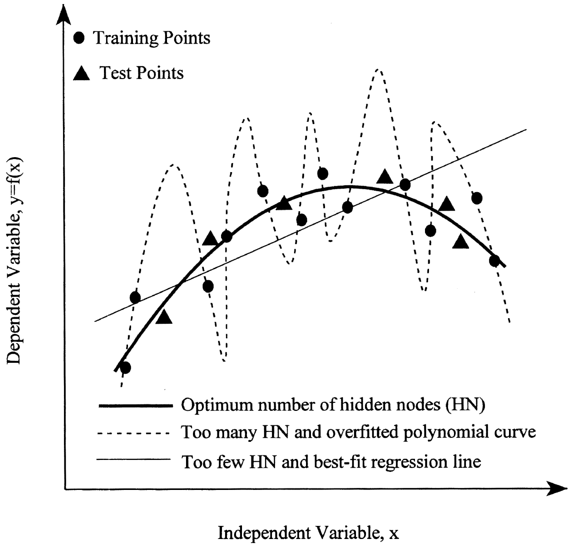


Fig. 9. Effect of hidden layer size on network generalization.

10. Lachtermacher and Fuller (1995) suggest that NHN for a one-output ANN with no biases be determined from $0.11N_{\mathrm{TRN}} \leq \mathrm{NHN}(N_{\mathrm{INP}} + 1) \leq 0.30N_{\mathrm{TRN}}$. Masters (1994) suggests that the ANN architecture should resemble a pyramid with $\mathrm{NHN} \approx (N_{\mathrm{INP}} \cdot N_{\mathrm{OUT}})^{1/2}$. Hecht-Nielsen (1990) used the Kolmogrov theorem to prove that $\mathrm{NHN} \leq N_{\mathrm{INP}} + 1$. Upadhaya and Eryureka (1992) related NHN to N_{TRN} (via the total number of weights, N_{w}) according to $N_{\mathrm{w}} = N_{\mathrm{TRN}} \log_2(N_{\mathrm{TRN}})$, and Widrow and Lehr (1990) according to $(N_{\mathrm{w}}/N_{\mathrm{OUT}}) \leq N_{\mathrm{TRN}} \leq (N_{\mathrm{w}}/N_{\mathrm{OUT}}) \log_2(N_{\mathrm{w}}/N_{\mathrm{OUT}})$.

Facing exotic problems such as those with high nonlinearity and hysteresis (e.g., Basheer, 1998, 2000) normally forces the modeler to try networks with hidden layers that may not conform to any of these rules of thumb. The most popular approach to finding the optimal number of hidden nodes is by trial and error with one of the above rules as starting point. Another way is to begin with a small number of hidden nodes and build on as needed to meet the model accuracy demand. Again, the cross-validation technique for determining the proper size of the hidden layer involves monitoring the error on both the training and test subsets in a way similar to that used to stop training (Fig. 8). Among other popular but more sophisticated techniques of optimizing network size are the growing and pruning methods (Sietsma and Dow, 1988).

11.13. Parameter optimization

As can be seen, BP training requires a good selection of values of several parameters, commonly through trial and error. Six parameters should not be set too high (large) or too low (small), and thus should be optimized or carefully selected. Table 1 lists these parameters and their effect on both learning convergence and overall network performance.

12. BP learning troubleshooting

In ANN development, the error on both the training and test subsets is monitored per training cycle and increase in the number of hidden nodes, as described in Fig. 8. During training by BP, the modeler may be faced with a number of situations that could impede the process of developing a proper network. A number of common situations are presented below with possible causes and potential remedies.

(i) The error on both training and test subsets is decreasing monotonically but slowly. This is a healthy sign indicating that the system is learning properly and improving with time with regard to both the trained data and the generalization on test data. In order to speed up learning, the learning rate or momentum may be increased.

(ii) The learning rate has been increased to speed up the learning process but the error change is seen to fluctuate throughout the learning cycles. This may indicate that the system is unstable and learning is accompanied by forgetting (Henseler, 1995). Reducing the learning rate and retraining is a possible remedy.

(iii) The error on the training subset has decreased to nearly zero but the error on the test subset only decreased to a high level and almost fattened out thereafter. This indicates that all that could be learnt from the data has been captured by the network. There is no advantage from

Table 1
Effect of extreme values of design parameters on training convergence and network generalization

Design parameter	Too high or too large	Too low or too small
Number of hidden nodes (NHN)	Overfitting ANN (no generalization)	Underfitting (ANN unable to obtain the underlying rules embedded in the data)
Learning rate (η)	Unstable ANN (weights) that oscillates about the optimal solution	Slow training
Momentum coefficient (μ)	Reduces risk of local minima. Speeds up training. Increased risk of overshooting the solution (instability)	Suppresses effect of momentum leading to increased risk of potential entrapment in local minima. Slows training
Number of training cycles	Good recalling ANN (i.e., ANN memorization of data) and bad generalization to untrained data	Produces ANN that is incapable of representing the data
Size of training subset (N_{TRN})	ANN with good recalling and generalization	ANN unable to fully explain the the problem. ANN with limited or bad generalization
Size of test subset (N_{TST})	Ability to confirm ANN generalization capability	Inadequate confirmation of ANN generalization capability

training any further in an attempt to reduce the error on the test subset.

(iv) The error on the training subset decreased to almost zero, but the error on the test subset started to increase after an initial decline. This indicates that the network has started to memorize the data, and its generalization capability has deteriorated. Check the point (i.e. NHN and/or training cycles) of minimum error on the test subset error curve, and retrain the network to that point.

(v) The error on the training subset has declined to almost zero but that of the test subset is considerably high and did not change since the beginning of training. This could be a sign of unrepresentative test data such as examples from outside the problem domain or with erroneous data. The test subset should be inspected and any bad example removed or examples from outside the training data domain transferred to the training subset.

(vi) The error on the training data declined at the beginning of training but is still high even after a large number of training cycles. This behavior is common when a small learning rate is used. Increasing the training rate may help alleviate the problem. However, if the problem persists or new unstable behavior is observed, the presence of inconsistent examples (bad/contradictory) in the training subset is suspected. Thorough statistical analysis of the outliers in the data may be needed to cleanse the training subset. In some situations, a different data representation scheme or increasing the number of hidden nodes may be needed.

(vii) The error on both the training and test subsets dropped to certain levels but did not change thereafter. If these levels are acceptable then the network has learnt what it is supposed to learn. New training should be conducted up to the onset of this steady behavior and the final network architecture is saved. If the levels are unacceptable, new hidden nodes may be added gradually followed by retraining.

(viii) The errors on both the training and test subsets are considerably high and are not changing with training cycle. This may indicate that the search is stuck in a local minimum or the problem as presented has no solution. More often, however, what seemed to have been a local minimum could actually be that the weights have become large enough to run the neurons into saturation where the derivatives are close to zero. Changing the weight initialization method may help remedy this problem. If it is a local minimum problem, increasing the momentum may help to escape it.

13. Application

13.1. General

A great portion of ANN applications in microbiology is in the analysis of pyrolysis mass spectra for microorganism identification (e.g., Chun et al., 1993a,b; Giacomini et al., 1997; Timmins and Goodacre, 1997), and the analysis of food systems (e.g., Goodacre et al., 1993; Goodacre, 1997). Results of these studies indicate that ANNs are more robust and rapid than traditional techniques. Carson et al. (1995) utilized ANNs for the automated identification of *Escherichia coli* O157:H7 restriction patterns, and indicated higher accuracy and cost effectiveness of ANNs over conventional sequential processor systems. Using GC data, Vallejo-Cordoba et al. (1995) applied ANNs for the determination of the shelf life of milk, and Horimoto et al. (1997) used ANNs for the classification of microbial defects in milk. Both studies showed superior performance of ANNs over least square regression and principal component analyses. Ramos-Nino et al. (1997) compared the performance of ANNs to multiple linear regression in developing QSARs for modeling the effect of benzoic and cinnamic acids on the inhibition of *Listeria monocytogenes*, and found better predictions by ANNs. Hajmeer et al. (1996, 1997, 1998) developed ANNs for predicting the growth of *Saccharomyces cerevisiae*, *Shigella flexneri* and *E. coli* O157:H7, which outperformed the corresponding traditional response surface models.

13.2. Modeling growth curves

As an illustration of ANN development, the growth curves of *S. flexneri* will be modeled using BP ANNs. The data were obtained from Zaika et al. (1994) which involved generating growth curves for 124 combinations of temperature, $T = 12–27°C$, pH 5.5–7.5, $[NaCl] = 0.5–4.0\%$, and $[NaNO_2] = 0–1000$ ppm. Zaika et al. (1994) fitted the experimental curves (log count, L vs. time, t) to the Gompertz equation expressed as $L(t) = A + C\exp\{-\exp[-B(t-M)]\}$, where A and C are, respectively, the logarithms of the microorganisms' initial level and asymptotic level as $t \to \infty$, M is the time corresponding to the maximum absolute growth rate, and B is the relative growth rate at $t = M$. The parameters C, B, and M are experiment specifics, thus they depend on A and the operating conditions. For each experiment, Zaika et al. (1994) backcalculated the Gompertz parameters (C, B, M), and the specific growth parameters were determined. Finally, the obtained growth parameters were regressed against the corresponding operating conditions (T, pH, [NaCl], and $[NaNO_2]$) and correlation equations (response surface models) were developed. Hajmeer et al. (1997) developed ANN models as alternatives to these equations, which improved prediction accuracy. Since Zaika et al. (1994) provided only the obtained Gompertz parameters (A, C, B, M) but not the experimental growth curves, the approximate curves were generated using the Gompertz equation. For ANN development, we selected 12 curves with $[NaCl] = 2.5\%$ and $[NaNO_2] = 0$ ppm and thus will be eliminated from the model. The curves are displayed in Fig. 10a with $T = 15–37°C$ and pH 5.0–7.5. As can be seen, the peak counts for the 12 curves are within 2 logs, and the time to peak varies from 10 to 1000 h.

One advantage of using ANNs for modeling growth curves is their ability to incorporate all of the operating parameters in one model. In comparison, the inclusion of such parameters in a closed-form mathematical expression such as the Gompertz equation seems to be extremely difficult. The time-dependent growth model may be expressed as $L = f(A,T,\text{pH},t)$, where f refers to a mathematical function representing the underlying process. The purpose of ANN modeling is to design an ANN that approximates, as closely as possible, the function f. The input layer of the ANN will contain the independent variables A, T, pH and t, and the output node will be L. A single hidden layer was used in developing the ANN, and the steps in the ANN development along with some results are described below.

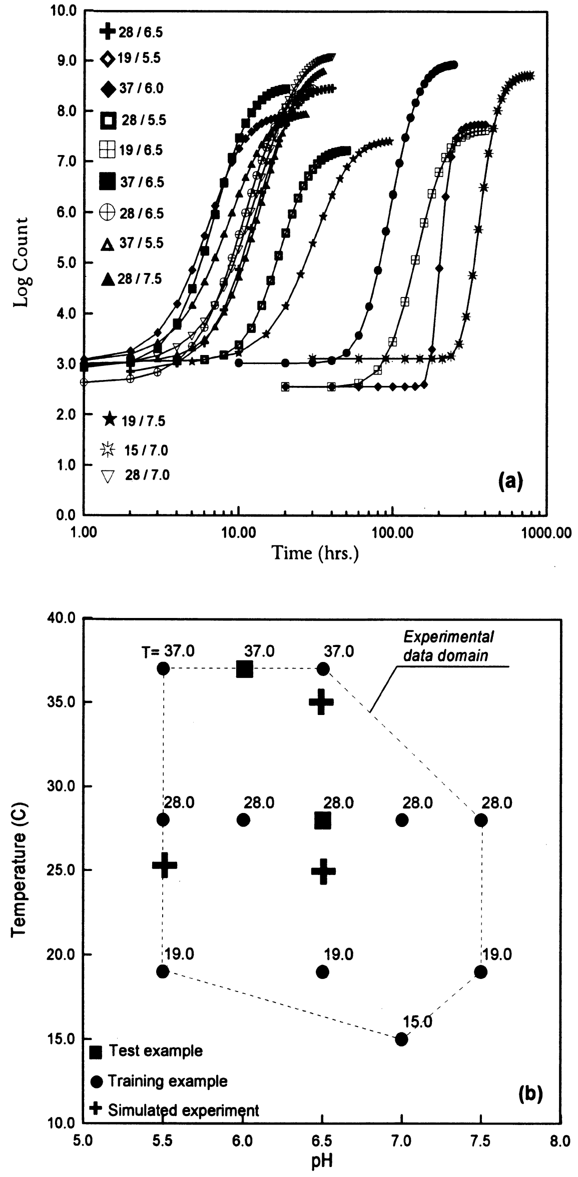


Fig. 10. (a) The 12 growth curves of *S. flexneri* used in ANN development. (b) Distribution of all the growth curves and selection of the training, test, and validation curves.

(i) The input and output data were preprocessed by normalization into a uniform range between 0.05 and 0.95 using Eq. (10). The database was split into a training subset and a test subset. To choose the test data from within the training data domain, the 12 (pH, T) pairs were placed on a pH–T plot as shown in Fig. 10b. The dashed line represents the experimental data domain, and thus any test curve should be selected from within this region. Because of the small number of example curves, only two curves (17% of the available curves) were used for testing [(pH, T) = (6.0, 37.0), (6.5, 28.0)]. The remaining 10 curves were used for training. The data were encoded such that each point on the curve (Fig. 10a) is represented by a vector whose input part is comprised of A, T, pH, and t, and whose output is the corresponding L. This yielded a total of 262 training data points and 52 test points.

(ii) A variety of learning rates and momenta were tried but the systems oscillated, and for high values of these parameters it never converged. We finally used $\eta = 0.10$ and $\mu = 0.90$. Training was stopped whenever the error was ≤ 0.001 on each training example or 5000 iterations occurred first. If the 5000 iterations governed, the network was expanded by adding one more hidden node.

(iii) The error was computed in three different ways: a combined SSE (training plus test data) was used to select the best performing network, R^2, and the mean of the absolute values of the relative error (denoted by MARE) expressed in percent (Hajmeer et al., 1998). Fig. 11a shows the variation of the combined SSE as a function of both the number of training cycles and hidden nodes. It can be seen that the network was unable to learn the underlying process with one or two hidden nodes; however, the addition of a third node resulted in a significant drop in SSE ($\approx$10-fold). This is also demonstrated in Fig. 11b using R^2 on the training data (R^2 increased from 0.4 to 0.9). The optimal network occurred at 10 hidden nodes (network denoted by 4-10-1 to refer to the number of nodes in each layer) trained to 2800 cycles. For this ANN, the prediction accuracy measures were: R^2 (training data) = 0.982 and R^2 (test data) = 0.992, MARE (training data) = 5.62% and MARE (test data) = 3.39%. SSE = 21.25 for 262 training data points (i.e. 0.081 per point), and 4.27 for 52 test data points (i.e. 0.082 per point).

(iv) The 4-10-1 ANN was used to simulate the training (i.e. recall) and test (i.e. prediction) growth curves. Fig. 12a shows a sample of

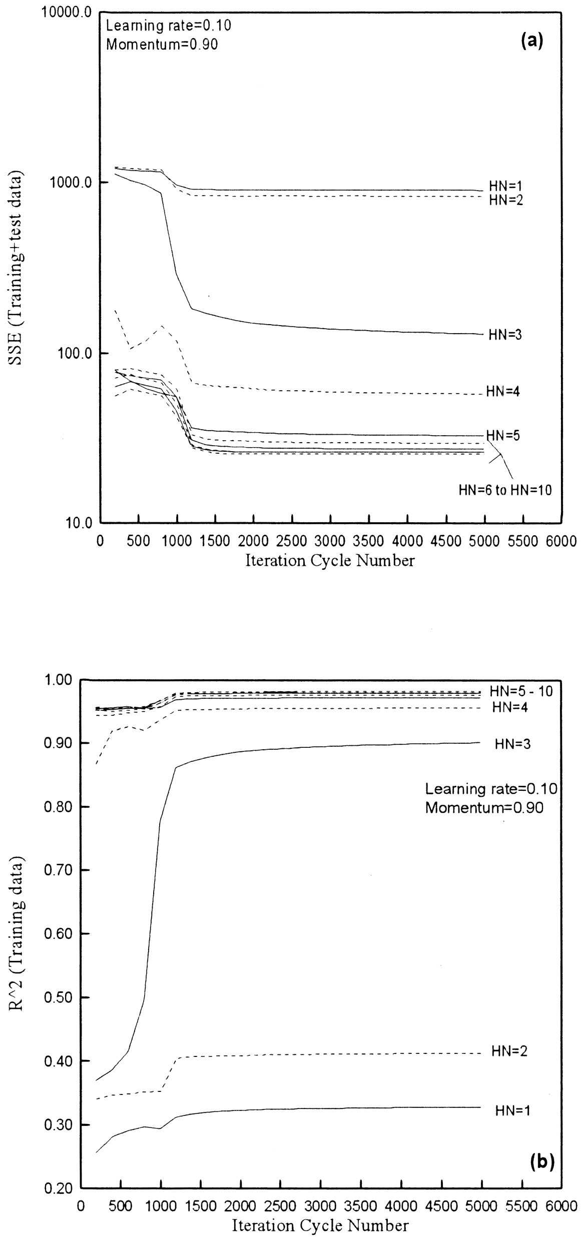


Fig. 11. (a) Effect of the number of training cycles and hidden layer size on the combined training and testing SSE. (b) Effect of the number of training cycles and hidden layer size on training R^2.

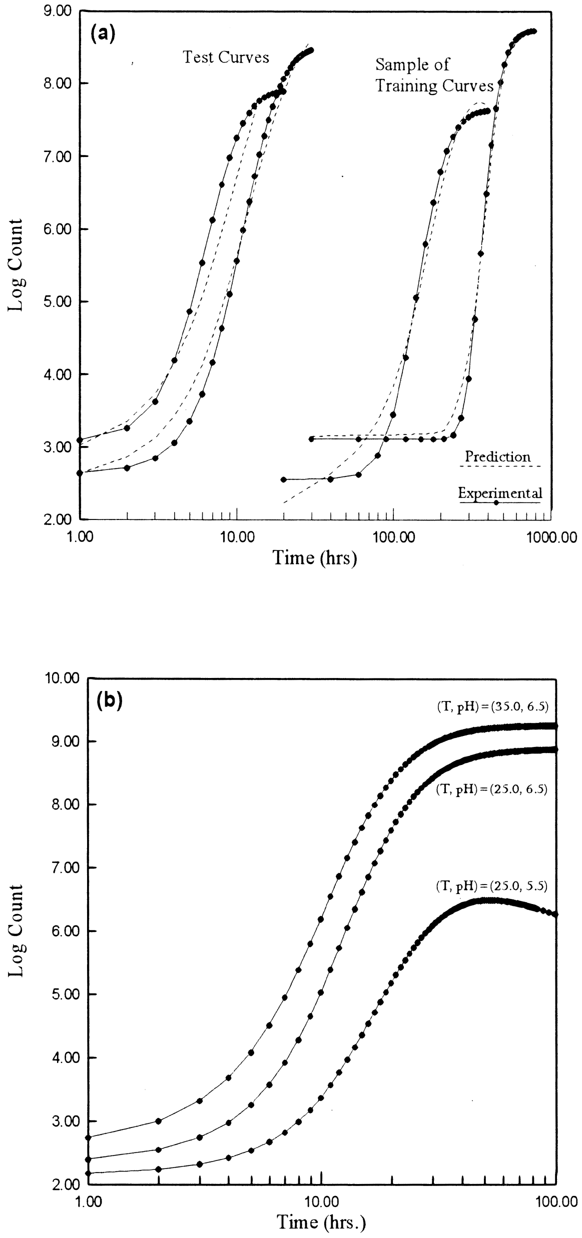


Fig. 12. (a) ANN prediction for a sample of training and test curves. (b) Validation of ANN accuracy using three new temperature–pH combinations.

simulated curves in comparison with their experimental counterparts. It is evident that the network was able to reproduce the training curves and predict the two unlearned test curves reasonably well. This implies that the network has learnt the underlying process.

(v) One important feature of the obtained model is its ability to simulate other growth curves for conditions not previously tested experimentally. Besides prediction, this can be used as a means to study the sensitivity of the operating parameters to the model output. As shown in Fig. 10b, we selected three new (pH, T) combinations, (5.5, 25.0), (6.5, 25.0), and (6.5, 35.0), to study the pH effect on growth at constant temperature ($T = 25.0$°C) and the temperature effect at a constant pH of 6.5. The simulations shown in Fig. 12b were obtained by running the 4-10-1 ANN with $A = 3.00$ for up to 100 h. It is seen that increasing the pH would increase the peak count by 3 logs, whereas increasing the temperature from 25 to 35°C would slightly enhance the growth. The above two trends are logical and compare well with the trend displayed in Fig. 10a. Finally, since the network is an empirical model, it is essential that it be used within the ranges of data used in its development ($T = 15$–37°C, pH 5.0–7.5, $A = 2.56$–3.11, $t = 0$–1000 h).

14. Concluding remarks

The remarkable information processing capabilities of ANNs and their ability to learn from examples make them efficient problem-solving paradigms. From a bird's eye perspective, an historical summary of the evolution of the field of neurocomputing was presented along with a review of the basic issues pertaining to ANN-based computing and ANN design. A generalized methodology for developing ANN projects from the early stages of data acquisition to the latest stages of utilizing the model to derive useful information was also proposed and discussed. An application of BPANNs for developing time-dependent growth curves for *S. flexneri* in a salty environment and under the effect of temperature and pH is presented for illustration purposes.

The increased utilization of ANNs is linked to several features they possess, namely (i) the ability to recognize and learn the underlying relations between input and output without explicit physical consideration, regardless of the problem's dimensionality and system nonlinearity, and (ii) the high tolerance to data containing noise and measurement errors due to distributed processing within the network. ANNs also have limitations that should not be overlooked. These include (i) ANNs' success depends on both the quality and quantity of the data, (ii) a lack of clear rules or fixed guidelines for optimal ANN architecture design, (iii) a lack of physical concepts and relations, and (iv) the inability to explain in a comprehensible form the process through which a given decision (answer) was made by the ANN (i.e., ANNs criticized for being black boxes).

ANNs are not a panacea to all real-world problems; for that, other traditional (non-neural) techniques are powerful in their own ways. Hybridizing ANNs with conventional approaches such as expert systems can yield stronger computational paradigms for solving complex and computationally expensive problems. Recently, ANNs have attracted the attention of the microbiology community, particularly in the area of pyrolysis mass spectrometry and microbial growth in food systems. The interest in ANNs will continue to grow and the application of such a technique to modeling a larger class of problems in microbiology will be on the rise.

References

Anderson, J.A., Rosenfeld, E., 1988. Neurocomputing: Foundations of Research. MIT Press, Cambridge, MA.

ASCE, 2000. Artificial neural networks in hydrology. I. Preliminary concepts. J. Hydro. Eng. ASCE 5, 115–123.

Attoh-Okine, N., Basheer, I., Chen, D.-H., 1999. Use of artificial neural networks in geomechanical and pavement systems. Trans. Res. Board, Circular E-C012.

Barron, A.R., 1993. Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans. Inform. Theory 39 (3), 930–945.

Basheer, I.A., 1998. Neuromechanistic-based modeling and simulation of constitutive behavior of fine-grained soils. PhD Dissertation, Kansas State University, 435 pp.

Basheer, I., 2000. Selection of methodology for modeling hysteresis behavior of soils using neural networks. J. Comput.-aided Civil Infrastruct. Eng. 5 (6), 445–463.

Baum, E., Haussler, D., 1989. What size net gives valid generalization? Neural Computation 1, 151–160.

Bishop, C., 1995. Neural Networks For Pattern Recognition. Oxford University Press, Oxford.

Carpenter, G.A., Grossberg, S., 1988. The ART of adaptive pattern recognition by a self-organizing neural network. Computer March, 77–88.

Carpenter, G.A., Grossberg, S., 1987. ART2: self-organization of stable category recognition codes for analog input patterns. Appl. Opt. 26, 4914–4930.

Carson, C.A., Keller, J.M., McAdoo, K.K., Wang, D., Higgins, B., Bailey, C.W., Thorne, J.G., Payne, B.J., Skala, M., Hahn, A.W., 1995. *Escherichia coli* O157:H7 restriction pattern recognition by artificial neural network. J. Clin. Microbiol. 33, 2894–2898.

Cheng, B., Titterington, D.M., 1994. Neural networks: a review from a statistical perspective. Statist. Sci. 9, 2–54.

Chun, J., Atalan, E., Kim, S.B., Kim, H.J., Hamid, M.E., Trujillo, M.E., Magee, J.G., Manfio, G.P., Ward, A.C., Goodfellow, M., 1993a. Rapid identification of *Streptomyces* by artificial neural network analysis of pyrolysis mass spectra. FEMS Microbiol. Lett. 114, 115–119.

Chun, J., Atalan, E., Ward, A.C., Goodfellow, M., 1993b. Artificial neural network analysis of pyrolysis mass spectrometric data in the identification of *Streptomyces* strains. FEMS Microbiol. Lett. 107, 321–325.

Dowla, F.U., Rogers, L.L., 1995. Solving Problems in Environmental Engineering and Geosciences With Artificial Neural Networks. MIT Press, Cambridge, MA.

Eaton, H.A.C., Olivier, T.L., 1992. Learning coefficient dependence on training set size. Neural Networks 5, 283–288.

Fahlman, S.E., 1988. An empirical study of learning speed in backpropagation. Technical Report, CMU-CS-88-162, Carnegie-Mellon University.

Fu, L., 1995. Neural Networks in Computer Intelligence. McGraw-Hill, New York.

Garth, A.D.N., Rollins, D.K., Zhu, J., Chen, V.C.P. et al., 1996. Evaluation of model discrimination techniques in artificial neural networks with application to grain drying. In: Dagli, C.H. et al. (Eds.). Artificial Neural Networks In Engineering, ANNIE, Vol. 6, pp. 939–950.

Geeraerd, A.H., Herremans, C.H., Cenens, C., Van Impe, J.F., 1998. Application of artificial neural networks as a nonlinear modular modeling technique to describe the bacterial growth in chilled food products. Int. J. Food Microbiol. 44, 49–68.

Giacomini, M., Ruggiero, C., Bertone, S., Calegari, L., 1997. Artificial neural network identification of heterotrophic marine bacteria based on their fatty-acid composition. IEEE Trans. Biomed. Eng. 44, 1185–1191.

Goodacre, R., Howell, S.A., Noble, W.C., Neal, M.J., 1994. Sub-species discrimination using pyrolysis mass spectrometry and self-organizing neural networks of *Propionibacterium acnes* isolated from normal human skin. Zbl. Bakt. 22, 124–136.

Goodacre, R., 1997. Use of pyrolysis mass spectrometry with supervised learning for the assessment of the adulteration of milk of different species. Appl. Spectrosc. 51, 1144–1153.

Goodacre, R., Kell, D.B., Bianchi, G., 1993. Rapid assessment of olive oil adulteration using pyrolysis mass spectrometry and artificial neural networks. J. Sci. Food Agric. 63, 297–307.

Hajmeer, M.N., Basheer, I.A., Najjar, Y.M., 1996. The growth of *Escherichia coli* O157:H7 — a backpropagation neural network approach. In: Dagli, C.H. et al. (Eds.), Proceedings ANNIE'96 — Intelligent Engineering Systems Through Artificial Neural Networks. ASME Press, pp. 635–640.

Hajmeer, M.N., Basheer, I.A., Najjar, Y.M., 1997. Computational neural networks for predictive microbiology. II. Application to microbial growth. Int. J. Food Microbiol. 34, 51–66.

Hajmeer, M.N., Basheer, I.A., Fung, D.Y.C., Marsden, J.L., 1998. A nonlinear response surface model based on artificial neural networks for growth of *Saccharomyces cerevisiae*. J. Rapid Methods Automation Microbiol. 6, 103–118.

Hajmeer, M.N., Basheer, I.A., Fung, D.Y.C., Marsden, J.L., 2000. New approach for modeling generalized microbial growth curves using artificial neural networks (under review).

Han, J., Moraga, C., Sinne, S., 1996. Optimization of feedforward neural networks. Eng. Appl. Artif. Intell. 9 (2), 109–119.

Hanson, S.J., 1995. Backpropagation: some comments and variations. In: Rumelhart, D.E., Yves, C. (Eds.), Backpropagation: Theory, Architecture, and Applications. Lawrence Erlbaum, NJ, pp. 237–271.

Hassoun, M.H., 1995. Fundamentals of Artificial Neural Networks. MIT Press, Cambridge, MA.

Haykin, S., 1994. Neural Networks: A Comprehensive Foundation. Macmillan, New York.

Hebb, D.O., 1949. The Organization of Behavior. Wiley, New York.

Hecht-Nielsen, R., 1988. Applications of counterpropagation networks. Neural Networks 1, 131–139.

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

Henseler, J., 1995. Backpropagation. In: Braspenning, P.J. et al. (Eds.), Artificial Neural Networks, An Introduction to ANN Theory and Practice. Lecture Notes in Computer Science. Springer, NY, pp. 37–66.

Hertz, J., Krogh, A., Palmer, R.G., 1991. Introduction to the Theory of Neural Computation. Addison-Wesley, Reading, MA.

Hoffman, R., 1987. The problem of extracting the knowledge of experts from the perspective of experimental psychology. AI Mag. 8, 53–67.

Hopfield, J.J., 1984. Neurons with graded response have collective computational properties like those of two-state neurons. Proc. Natl. Acad. Sci. 81, 3088–3092.

Hopfield, J.J., Tank, D.W., 1986. Computing with neural circuits: a model. Science 233, 625–633.

Horimoto, Y., Karoline, L., Nakai, S., 1997. Classification of microbial defects in milk using a dynamic headspace gas chromatograph and computer-aided data processing. 2. Artificial neural networks, partial least-squares regression analysis, and principal component regression analysis. J. Agric. Food Chem. 45, 743–747.

Hudson, P., Postma, E., 1995. Choosing and using a neural net. In: Braspenning, P.J. et al. (Eds.), Artificial Neural Networks, An Introduction to ANN Theory and Practice. Lecture Notes in Computer Science. Springer, NY, pp. 273–287.

Jadid, M.N., Fairbairn, D.R., 1996. Predicting moment-curvature parameters from experimental data. Eng. Appl. Artif. Intell. 9 (3), 309–319.

Jain, A.K., Mao, J., Mohiuddin, K.M., 1996. Artificial neural networks: a tutorial. Comput. IEEE March, 31–44.

Jonsson, A., Winquist, F., Schunrer, J., Sundgren, H., Lundston, I., 1997. Electronic nose for microbial classification of grains. Int. J. Food Microbiol. 35, 187–193.

Kandel, A., Langholz, G. (Eds.), 1992. Hybrid Architectures For Intelligent Systems. CRC Press, Boca Raton, FL.

Kerber, R., 1992. ChiMerge: discretization of numeric attributes.

In: AAAI-92, Proceedings of the 9th National Conference on AI. MIT Press, Cambridge, MA, pp. 123–128.

Kohonen, T., 1989. Self-organization and Associative Memory, 3rd Edition. Springer, New York.

Lachtermacher, G., Fuller, J.D., 1995. Backpropagation in time-series forecasting. J. Forecasting 14, 381–393.

Lakshmanan, V., 1997. Detecting rare signatures. In: Dagli, C.H. et al. (Eds.). Artificial Neural Networks in Engineering, ANNIE, Vol. 7. ASME Press, NY, pp. 521–526.

Lee, Y., Oh, S.H., Kim, M., 1991. The effect of initial weights on premature saturation in backpropagation learning. In: Proceedings of an International Joint Conference on Neural Networks, Seattle, WA, pp. 765–770.

Li, G., Alnuweiri, H., Wu, W., 1993. Acceleration of backpropagation through initial weight pre-training with Delta rule. In: Proceedings of an International Joint Conference on Neural Networks, San Francisco, CA, pp. 580–585.

Lippmann, R.P., 1987. An introduction to computing with neural nets. IEEE ASSP Mag. 4, 4–22.

Looney, C.G., 1996. Advances in feedforward neural networks: demystifying knowledge acquiring black boxes. IEEE Trans. Knowledge Data Eng. 8 (2), 211–226.

Maren, A.J., 1991. A logical topology of neural networks. In: Proceedings of the Second Workshop on Neural Networks, WNN-AIND 91.

Masters, T., 1994. Practical Neural Network Recipes in C++. Academic Press, Boston, MA.

McCulloh, W.S., Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. 5, 115–133.

Minsky, M., Pappert, S., 1969. Perceptrons. MIT Press, Cambridge, MA.

Moody, J., Yarvin, N., 1992. Networks with learned unit response functions. In: Moody, J. et al. (Eds.). Advances in Neural Information Processing Systems, Vol. 4. Morgan Kaufmann, San Mateo, CA, pp. 1048–1055.

Najjar, Y., Basheer, I., Hajmeer, M., 1997. Computational neural networks for predictive microbiology. I. Methodology. Int. J. Food Microbiol. 34, 27–49.

Nelson, M., Illingworth, W.T., 1990. A Practical Guide To Neural Nets. Addison-Wesley, Reading, MA.

Nguyen, D., Widrow, B., 1990. Improving the learning speed of two-layer neural networks by choosing initial values of the adaptive weights. In: Proceedings of an IEEE International Joint Conference on Neural Networks, San Diego, CA, pp. 21–26.

Ni, H., Gunasekaran, S., 1998. Food quality prediction with neural networks. Food Technol. 5, 60–65.

Pal, S.K., Srimani, P.K., 1996. Neurocomputing: motivation, models, and hybridization. Computer March, 24–28.

Pham, D.T., 1994. Neural networks in engineering. In: Rzevski, G. et al. (Eds.), Applications of Artificial Intelligence in Engineering IX, AIENG/94, Proceedings of the 9th International Conference. Computational Mechanics Publications, Southampton, pp. 3–36.

Plant, R.E., Stone, N.D., 1991. Knowledge-based Systems in Agriculture. McGraw-Hill, New York.

Ramos-Nino, M.E., Ramirez-Rodriguez, C.A., Clifford, M.N., Adams, M.R., 1997. A comparison of quantitative structure–activity relationships for the effect of benzoic and cinnamic acids on *Listeria monocytogenes* using multiple linear regression, artificial neural network and fuzzy systems. J. Appl. Microbiol. 82, 168–176.

Rizzo, D.M., Dougherty, D.E., 1994. Characterization of aquifer properties using artificial neural networks: neural kriging. Water Res. 30 (2), 483–497.

Rosenblatt, R., 1962. Principles of Neurodynamics. Spartan Books, New York.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning internal representation by error propagation. In: Rumelhart, D.E., McClleland, J.L. (Eds.). Parallel Distributed Processing: Exploration in the Microstructure of Cognition, Vol. 1. MIT Press, Cambridge, MA, Chapter 8.

Rumelhart, D.E., Durbin, R., Golden, R., Chauvin, Y., 1995. Backpropagation: the basic theory. In: Rumelhart, D.E., Yves, C. (Eds.), Backpropagation: Theory, Architecture, and Applications. Lawrence Erlbaum, NJ, pp. 1–34.

Schalkoff, R.J., 1997. Artificial Neural Networks. McGraw-Hill, New York.

Schmidt, W., Raudys, S., Kraaijveld, M., Skurikhina, M., Duin, R., 1993. Initialization, backpropagation and generalization of feed-forward classifiers. In: Proceeding of the IEEE International Conference on Neural Networks, pp. 598–604.

Sietsma, J., Dow, R.J.F., 1988. Neural net pruning — why and how. IEEE Int. Conf. Neural Networks 1, 325–333.

Simpson, P.K., 1990. Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations. Pergamon Press, New York.

Specht, D.F., 1991. A general regression neural network. IEEE Trans. Neural Networks 2 (6), 568–576.

Stein, R., 1993. Selecting data for neural networks. AI Expert 2, 42–47.

Sui, D.Z., 1992. An initial investigation of integrating neural networks with GIS for spatial decision making. In: Proceedings GIS/LIS 92, San Jose, CA.

Sun, X., Wang, Q., Zhu, D., 1997. Function approximation and prediction: a comparative study with neural networks and regression models. In: Proceedings of Intelligent Engineering Systems Through Artificial Neural Networks, ANNIE, pp. 70–84.

Swingler, K., 1996. Applying Neural Networks: A Practical Guide. Academic Press, New York.

Timmins, E.M., Goodacre, R., 1997. Rapid quantitative analysis of binary mixtures of *Escherichia coli* strains using pyrolysis mass spectrometry with multivariate calibration and artificial neural networks. J. Appl. Microbiol. 83, 208–218.

Twomey, J., Smith, A., 1997. Validation and verification. In: Kartam, N., Flood, I. (Eds.), Artificial Neural Networks for Civil Engineers: Fundamentals and Applications. ASCE, pp. 44–64.

Upadhaya, B., Eryureka, E., 1992. Application of neural network for sensory validation and plant monitoring. Neural Technol. 97, 170–176.

Vallejo-Cordoba, B., Arteaga, G.E., Nakai, S., 1995. Predicting milk shelf-life based on artificial neural networks and head-space gas chromatographic data. J. Food Sci. 60, 885–888.

van Rooij, A., Jain, L., Johnson, R., 1996. Neural Network Training Using Genetic Algorithms. World Scientific, Singapore.

von Neuman, J., 1958. The Computer and the Brain. MIT Press, Cambridge, MA.

Werbos, P.J., 1974. Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD Thesis, Harvard University.

White, H., 1989. Learning in artificial neural networks: a statistical perspective. Neural Computation 1, 425–464.

White, H., 1990. Connectionist nonparametric regression — multilayer feedforward networks can learn arbitrary mappings. Neural Networks 3, 535–549.

Widrow, B., Lehr, M.A., 1990. 30 years of adaptive neural networks: perceptron, Madaline, and backpropagation. Proc. IEEE 78 (9), 1415–1442.

Wythoff, B.J., 1993. Backpropagation neural networks: a tutorial. Chemometr. Intell. Lab. Syst. 18, 115–155.

Yu, X., Loh, N., Miller, W., 1993. A new acceleration technique for the backpropagation algorithm. In: Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA, pp. 1157–1161.

Zaika, L.L., Moulden, E., Weimer, L., Phillips, J.G., Buchanan, R.L., 1994. Model for the combined effects of temperature, initial pH, sodium chloride and sodium nitrite concentrations on anaerobic growth of *Shigella flexneri*. Int. J. Food Microbiol. 23, 345–358.

Zupan, J., Gasteiger, J., 1991. Neural networks: a new method for solving chemical problems or just a passing phase? Anal. Chim. Acta 248, 1–30.

Zupan, J., Gasteiger, J., 1993. Neural Networks For Chemists: An Introduction. VCH, New York.