International Journal of Neural Systems, Vol. 3, No. 3 (1992) 253–290

RECENT ADVANCES ON TECHNIQUES OF STATIC FEEDFORWARD NETWORKS WITH SUPERVISED LEARNING

LEI XU
Department of Computer Science, Concordia University, Canada
Harvard Robotics Laboratory, Harvard University, USA
Department of Mathematics, Peking University, P. R. China

STAN KLASA
Department of Computer Science, Concordia University, Canada

and

ALAN YUILLE
Harvard Robotics Laboratory, Harvard University, USA

Received 23 March 1992

The rediscovery and popularization of the backpropagation training technique for multilayer perceptrons as well as the invention of the Boltzmann machine learning algorithm has given a new boost to the study on supervised learning networks. In recent years, besides widely spread applications and various further improvements of the classical backpropagation technique, many new supervised learning models, techniques as well as theories, have also been proposed in a vast number of publications. This paper tries to give a rather systematic review on the recent advances on supervised learning techniques and models for static feedforward networks. We summarize a great number of developments into four aspects:

(1) Various improvements and variants made on the classical backpropagation techniques for Multilayer (static) perceptron nets, for speeding up training, avoiding local minima, increasing the generalization ability as well as for many other interesting purposes.
(2) A number of other learning methods for training multilayer (static) perceptron, such as derivative estimation by perturbation, direct weight update by perturbation, genetic algorithms, recursive least square estimate and extended Kalman filters, linear programming, the policy of fixing one layer while updating another, constructing networks by converting decision tree classifiers and others.
(3) Various other feedforward models which are also able to implement function approximation, probability density estimation and classification, including various models of basis function expansion (e.g. radial basis functions, restricted coulomb energy, multivariate adaptive regression splines, trigonometric and polynomial bases, projection pursuit, basis function tree and many others) and several other supervised learning models.
(4) Models with complex structures, e.g. modular architecture, hierarchy architecture and others. Altogether, we try to give a global picture of the present state of supervised learning techniques (not including all the theoretical developments) for training static feedforward networks.

1. Introduction

The newly renascent neural network research has experienced an explosive period in recent years. During this period, not only numerous improvements and developments have been made on those models and methods developed in the 1960s and 1970s but also a number of new models and methods have been proposed. Moreover, many theoretical issues have been studied and clarified and tremendous practical applications have been explored. In particular, the study on *supervised learning neural networks* has developed drastically and has become perhaps the most popular branch in this field.

Triggered by Rosenblatt's Perceptron[181,182] and Widrow's Adaline,[239] the study of supervised learning nets was prosperous in the 1960s with a number of interesting algorithms proposed. Some of these algorithms like Perceptron,[181,182] Adaline[239]

and Ho-Kashyap[88] algorithms still survive vigorously nowadays. However, the research of this period was mainly concentrated on using one single neuron as a classifier which is only suitable for solving the linear separable problem.[44,148] Due to the lack of effective and workable training tools for multilayer neural networks and the shunting force of the increasingly heated waves of symbolic artificial intelligence in the same period and probably also partly due to Minsky and Papert's influential book,[148] the earlier wave of neural networks ceased around the end of the 1960s and the beginning of the 1970s.

The rediscovery and popularization of the backpropagation training technique for *multilayer feedforward networks* by Rumelhart, Hinton and Williams in 1986[183] as well as the invention of the Boltzmann machine learning algorithm[2] has caused a new boost to the study of supervised learning nets which has rapidly become a main stream of the current neural network renaissance. In recent years, besides widely spread applications and various further improvements of the classical backpropagation technique, many new supervised learning models, techniques as well as theories have also been proposed. Currently, there has already been a huge amount of publications about these topics in the literature. In order to pick up the main threads of recent developments, we first classify the results of a great diversity of learning techniques into categories and then define the scope of our survey in this paper.

Roughly speaking, various supervised learning nets can be grouped into two big categories: feedforward nets and recurrent nets. In a feedforward net, there is no circular or feedback information flow. By contrast in a recurrent net, there exists some circular or feedback (also called recurrent) information flow. The two categories of nets have salient differences in their dynamic characteristics. Recurrent nets are dynamic systems which can demonstrate dynamic behaviors of different complexities. Feedforward nets can be further divided into two subcategories. One consists of static nets which simply map static inputs into static outputs without any dynamic processing. A feedforward net is a static net if the temporal properties of all its neurons[a] and synapses[b] are ignored (i.e. these neurons and synapses are time-delayless devices). The other subcategory consists of feedforward nets in which some neurons and/or synapses have temporal properties (e.g. time delay and finite impulse response studied in Refs. 223, 47 and 22). These nets are again dynamic nets. In general, static nets are suitable for memory, association and recognition of spatial (or static) patterns while the dynamic nets are more suitable for memory, association and recognition of temporal sequences, and thus more suitable for the tasks of speech recognition, (Refs. 129, 3 and 100) robot control, (Refs. 106, 238) identification and control of other dynamic systems (Refs. 231, 154 and 137) as well as for studying the rhythmic and chaotic behaviors of biological systems (Refs. 6, 217, 36 and 251).

Due to space limitations, this survey will only focus on feedforward nets of the static subcategory, trained by supervised learning. In recent years, the studies of dynamic nets have also experienced an explosive development period, ranging from the studies of the classical quasistatic full connection symmetric net (i.e. the so-called Hopfield nets[84]) to stable dynamic nets (e.g. feedforward nets with time delays or finite pulse response properties (Refs. 223, 47 and 22), synchronous partially recurrent nets,[106,56] recurrent nets of real or continuous time (Refs. 241, 168, 230 etc.), oscillatory pattern generating nets[217] and even chaotic nets (Refs. 36, 224 and 251). Their development constitutes a rather comprehensive new subcategory. Thus we will put them aside and not review them here.

Presently in the literature, the developments on supervised learning static feedforward nets are scattered among many different publications. Although some of these developments have been reviewed in a number of the recently published survey papers (Refs. 12, 32, 45, 46, 139, 115, 87, 240, 5, 117, 170, and 75) and books (Refs. 74, 67, 116, 84, 156, 226, 185, 161, 111, 48, 83 and 20), there is still the lack of a complete and systematical review which covers all the major developments, especially those made in the recent two years. Here, according to our knowledge, we would like to summarize these major developments into the following five aspects.

- First, based on the classical backpropagation techniques for multilayer (static) perceptron nets, many improvements and variants have been made for speeding up training, avoiding local minima, increasing the generalization ability as well as for many other interesting purposes. E.g.
  (1) several heuristic techniques have been proposed for avoiding local minima and for speed-

[a] Or units or cells. These names are used interchangeably in this paper.
[b] Or connections or weights. They are also used interchangeably in this paper.

ing up learning (Refs. 119, 157, 135, 178, 248, 4, 216, 147, 228, 191, 58, 33, 204, 189, 41, 220, 202, 8, 86, 219, 38, 243 and 77) and other gradient techniques (Refs. 122, 141, 179, 21, 225 and 163) have been used to replace the simple steepest descent approach to increase convergence speed;

(2) a number of techniques are proposed for increasing the network's generalization ability (Refs. 183, 130, 206, 84, 124, 34, 35, 87, 80, 153, 101, 221, 215, 234, 13, 78, 85, 126, 76 and 19);

(3) in addition to the least square error criterion, several other error criteria[79] have been tested (Refs. 31, 81, 151, 152, 13, 71, 184, 223, 109, 200, 125, 82, 17, 87, 89, 57, 252, 94, 4, 141, 79, 11, 60, 108, 30, 58 and 127);

(4) some other variants (e.g. obtained by using other types of neurons) have been also studied (Refs. 211, 123, 235, 90, 200, 212, 243, 112, 97, 132, 70, 199 and 52).

- Second, the monopoly role of the backpropagation learning technique in the training of multilayer perceptrons has been shared by a number of other learning methods such as derivative estimation by perturbation (e.g. MRIII[7] and model free learning[50]), direct weight update by perturbation (e.g. local variation,[165] random optimization[9] and simulated annealing[49]), genetic algorithms (Refs. 236, 149, 237, 146 and 120), recursive least square estimate and extended Kalman filters (Refs. 205, 53, 172, 196, 190 and 118), linear programming,[198] the policy of fixing one layer while updating another (Refs. 1, 208, 72 and 66), constructing networks by converting decision tree classifiers (Refs. 194, 195, 27 and 218), and others (Refs. 114, 63 and 145). These methods may have one or more advantages over back propagation techniques, e.g. these advantages include better performance, fast learning, avoiding local minima, easy for VLSI fabrication etc.
- Third, many other models of static feedforward nets have been attracting the interests of more and more researchers. Examples of these models are various models which implement function approximation or probability density estimation by basis function expansion (Refs. 64, 95, 12, 65, 161, 10, 61, 173, 164, 175, 176, 192, 133, 98, 193, 209, 210, 37, 62, 171, 28, 169, 39, 68, 150, 177, 121, 158, 213, 186, 247, 166, 233, 107, 105 and 24), as well as a number of other unsupervised learning models (Refs. 117, 116, 155, 19, 99 and 249). Having some significant differences from the backpropagation approach, these models usually have closer relations with the theories of mathematical function approximation, statistical decision analysis and regression analysis.
- Fourth, instead of the sole preference of the conventional multilayer feedforward structure (i.e. the general purposed, distributed network structure fully connected between layers), in the recent two years, hierarchical, modularized and other specifically designed problem-dependent network structures have become increasingly popular (Refs. 103, 159, 244, 214, 187, 160, 23, 40, 16 and 25) because of the need for solving more complicated real problems. Moreover, these special structured nets can usually save a lot of training and storage cost and improve performance.
- Fifth, a number of theoretical issues have been studied and clarified. These results involve the universal approximation of continuous functions, best approximation ability, learnability, capability, generalization ability and the relations between these abilities to the number of layers in a network, the number of the needed neurons (or hidden neurons), as well as the number of training samples. These results can not only mathematically justify the uses of various neural nets but also guide the design, modification and application of these nets.

This paper is organized in such a way that from Secs. 2 to 4, the first four aspects above are surveyed in order. The issues of the fifth aspect will be addressed in a separate paper.

Due to the drastic rate of developments and the great number of publications in the literature, it is impossible to mention all the new developments in this single survey. Hence, we would like to say that our survey will only give a schematical outline (but by no means complete) with focus on the issues of methods and techniques. In particular, there are no applications reviewed here, although the tremendous and successful applications of *supervised learning nets* is an important feature in neural network research in recent years. Many application papers can be found in a number of journals and conference proceedings. The information not outlined in this paper can be found in several other good review papers such as Widrow[240] for the 30 years work of his group at Stanford University, Poggio[170] for regulation theory and RBF nets as well as their relations to several other kinds of nets, Barron *et al.*[12]

for the earlier results on theoretical tissues, Amari,[5] Kohonen,[117] Grossberg and Carpenter[32] for the surveys of their work over more than twenty years respectively. Moreover, the surveys on those previously developed and already quite well established methods and models can be found in several previous survey papers (Refs. 45, 46, 139, 115, 87 and 75) and a number of text books (Refs. 74, 67, 116, 84, 156, 226, 185, 161, 111, 48, 83 and 20).

2. Backpropagation Techniques: Improvements and Variants

2.1. *Multilayer perceptrons and backpropagation*

The backpropagation technique was proposed for training multilayers (static) perceptrons by Rumelhart *et al.*[183,c]

The architecture of the networks have the following three features:

- *Layered structure.*
  All the neurons are aligned into multiple layers.
- *Directed forward connections.*
  The neurons within the same layer have no connections while the neurons between two successive layers are fully connected (i.e. each neuron in a layer emits directed connections towards all the neurons of the suceeding layer).
- *Homogeneous neurons.*
  Every neuron in the whole net has the same type of sigmoid-summation function and every neuron in the same layer has the same fan-in number (i.e. the same number of connections towards every neuron of the same layer).

Assume that such a net has $L+1$ layers with the 0th (i.e. the bottom) layer being the input layer and the Lth (i.e. the top) layer being the output layer. Let $o_j^{(q)}$ denote the output of the jth neuron in the qth layer and $w_{jk}^{(q)}$ the connection synapses coming from the kth neuron in the $q-1$th layer, then we have

$$\begin{aligned} o_j^{(q)} &= \sigma(y_j^{(q)}) \\ y_j^{(q)} &= \sum_{k=1}^{n_{q-1}} w_{jk}^{(q)} o_k^{(q-1)}, \end{aligned} \tag{1}$$

where $y_j^{(q)}$ is called the activation level of the neuron, n_{q-1} is the number of neurons in the $q-1$th layer and $\sigma(y)$ is a sigmoid activation function given by

$$\begin{aligned} &\sigma(y) = 1/(1+e^{-ay}) \\ \text{or} \quad &\sigma(y) = (e^{ay}-e^{-ay})/(e^{ay}+e^{-ay}), \end{aligned} \tag{2}$$

where a is a slant parameter which is usually set at $a=1$.

When given an input $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ to the bottom layer, we have $\mathbf{o}^{(0)} = [o_1^{(0)}, \ldots, o_{n_0}^{(0)}] = \mathbf{x}$ and $n_0 = n$. The input goes up through the intermediate layers (usually called hidden layers) to the output layer to produce the outputs $f_i(x) = o_i^{(L)}$, $i = 1, \ldots, m$, where $m = n_L$. Therefore, as a whole, the net functions as a compound vector function $F(\mathbf{x}) = [f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})]$ given by

$$f_i(\mathbf{x}) = \sigma(\Sigma_k w_{ik}^{(L)} \sigma(\Sigma_r w_{kr}^{(L-1)} \sigma(\cdots \sigma(\Sigma_j w_{pj}^{(1)} x_j)))), \tag{3}$$

$$f_i(\mathbf{x}) = \Sigma_k w_{ik}^{(L)} \sigma(\Sigma_r w_{kr}^{(L-1)} \sigma(\cdots \sigma(\Sigma_j w_{pj}^{(1)} x_j))). \tag{4}$$

In fact, the function is determined by the value of each element in the weight set $\mathbf{W} = \{w_{ij}^{(q)} | w_{ij}^{(q)} \in R, q = 1, \ldots, L, i = 1, \ldots, n_{q-1}, j = 1, \ldots, n_q\}$. By varying the values taken by the set $\mathbf{W}$, the net is capable of implementing a set of functions

$$\begin{aligned} \Sigma = \{F : R^n \to R^m | F(\mathbf{x}) = F(\mathbf{x}, \mathbf{W}), \\ w_{ij}^{(q)} \in R, \quad \text{for all } w_{ij}^{(q)} \in \mathbf{W}\}. \end{aligned} \tag{5}$$

Given a set of training data $D_{tr} = \{(\mathbf{x}_p, \mathbf{d}_p), p = 1, \ldots, N_p\}$, with each $\mathbf{x}_p$ being an input vector and each $\mathbf{d}_p$ being the correspondent desired output vector as supervisor, where $\mathbf{x}_p$ can be either real (i.e. $\mathbf{x}_p \in R^n$) or binary (i.e. $\mathbf{x}_p \in [0,1]^n$); and $\mathbf{d}_p$ usually can be one of the three choices:

(1) real $\mathbf{d}_p \in R^m$ when the net is used for approximating a real function;
(2) binary $\mathbf{d}_p \in [0,1]^m$ when the net is used as a coder;
(3) a Vertex of the m dimensional unit hypercubic, i.e.

$$\begin{aligned} &\mathbf{d}_p \in V \subset [0,1]^m, \\ &V = \{[b_{i1}, \ldots, b_{im}]^t | b_{ik} = \delta_{ik} \\ &\qquad \text{for } i,k = 1, \ldots, m\}^{\text{d}} \end{aligned}$$

[c] In fact, the invention of the technique has a colorful history. It was invented independently several times (Refs. 29, 229 and 162) before it finally became popularized by Rumelhart *et al.* in 1986.

[d] $\delta_{ik} = 1$, if $i = k$; otherwise, $\delta_{ik} = 0$.

when the net is used as a classifier. The task of supervised learning is to let $\mathbf{W} = W$ with W being a set of specific values taken by each element of $\mathbf{W}$ in order to give a specific function

$$F(\mathbf{x}) = F(\mathbf{x}, \mathbf{W})|_{\mathbf{W}=W}\,, \tag{6}$$

such that under the least square error criterion $\epsilon^2 = ||\mathbf{d} - F(x)||^2$ the total error

$$\begin{aligned} E_2 &= \sum_{p=1}^{P} \epsilon_p^2\,, \quad \epsilon_p^2 = ||\epsilon_p||^2 \\ \epsilon_p &= \mathbf{d}_p - F(\mathbf{x}_p)\,, \end{aligned} \tag{7}$$

is minimized.

Thus, one reaches a typical nonlinear optimization problem. The backpropagation technique uses the *gradient descent* method for solving the problem, i.e. by iteratively adjusting each connection weight $w_{ji}^{(q)}$ by

$$\Delta w_{ji}^{(q)} = -\alpha \frac{\partial E_2}{\partial w_{ji}^{(q)}}\,, \tag{8}$$

where α is the step factor called the *learning rate.*

The key of the technique is to obtain the derivatives $\frac{\partial E_2}{\partial w_{ji}^{(q)}}$ for every weight in the net. When $p = L, F(\mathbf{x}) = [o_1^{(L)}, \ldots, o_m^{(L)}]$, it follows from Eq. (7) that the derivatives can be obtained by directly differentiating E_2; for $q < L$, by applying the *chain rule* on E_2, we can obtain

$$\frac{\partial E_2}{\partial w_{ji}^{(q)}} = \frac{\partial E_2}{\partial y_j^{(q)}} \frac{\partial y_j^{(q)}}{\partial w_{ji}^{(q)}} = \frac{\partial E_2}{\partial y_j^{(q)}} o_i^{(q-1)}\,, \tag{9}$$

$$\begin{aligned} \frac{\partial E_2}{\partial y_j^{(q)}} &= \sum_k \frac{\partial E_2}{\partial y_k^{(q+1)}} \frac{\partial y_k^{(q+1)}}{\partial y_j^{(q)}} \\ &= \sigma'(y_j^{(q)}) \sum_k \frac{\partial E_2}{\partial y_k^{(q+1)}} w_{kj}^{(q)}\,, \end{aligned} \tag{10}$$

i.e. the derivatives of the lower layer can be recursively obtained from the derivatives in its immediate upper layer. Or in other words the derivatives are propagated downwards by the simple formula Eq. (10) in the opposite direction to the upward information flow by Eq. (1), hence the name *backpropagation.*

The technique is really simple but it works, which makes it the most popular technique in the neural network field. However, it also has some disadvantages. Many variants and improvements have been made to overcome them, we will review the main results in the sequel.

2.2. *Techniques for avoiding local minima and speeding up training*

The first drawback of the above classical backpropagation is that gradient descent may sometimes be stuck at local minima. Although the problem is considered not to be serious in many cases, it does affect performances for some problems.[143,125] The following heuristics are often used to ease the problem (Refs. 41, 220, 202 and 8).

- *On-line updating.*
  In Eq. (8), the derivatives are calculated via the total error E_2 which is obtained after a batch of P patterns has been presented. The way is sometimes called *batch updating.* A simple version can be obtained by using the one-line error ϵ_p^2 to replace E_2 in Eqs. (8), (9) and (10), i.e. the weight updating is made once a pattern is presented. This way is called *on-line updating.* By choosing each pattern randomly from the training set, it will produce small random fluctuations which let the system get out of local minima, since these minima are usually not too deep.
- *Adding noise explicitly.*
  One way is to let Eq. (8) be replaced by $\Delta w_{ji}^{(q)} = -\alpha \frac{\partial E_2}{\partial w_{ji}^{(q)}} + n_t$ with n_t being Gaussian white noise of a small variance. Another way is to directly add such a n_t to each input pattern $\mathbf{x}_p$. The noise n_t can make fluctuations which let the system get out of local minima. However, the variance of n_t should be selected appropriately.
- *Annealing.*
  In the above cases, let the variance of n_t be initialized by a sufficiently large value and then gradually reduced to zero. This process is similar to *simulated annealing,*[113] which can usually get the global solution but make the learning quite slow. There is also another similar way. For *on-line updating,* after each updating of weights, we calculate the error change $\Delta\epsilon_p^2$. It may sometimes be positive (i.e. 'overshot' happens in the updating). In such a case, we accept the update by a Boltzman probability $p = \exp(-\Delta\epsilon_p^2/T)$; if the update fails to be accepted, then what has been learned in the update is abolished. Here the parameter T, called temperature, is initialized by a sufficiently large value and then gradually reduced to zero. This form of annealing is quite like that originally

proposed in Ref. 113.

The second and also more influential drawback of the classical backpropagation is its *slow learning speed.* There are four major reasons which cause the problem. Each of them, together with the existing remedies, are summarized in the sequel.

- *The gradient direction sometimes changes drastically.*
  For the error-surface of valleys with steep sides but a shallow slope along the valley floor, the gradient descent will oscillate across the valley for a very long time and trace a very inefficient long zigzag path. One commonly used remedy is to replace Eq. (8) by

$$\Delta w_{ji}^{(q)}(t+1) = -\alpha \frac{\partial E_2}{\partial w_{ji}^{(q)}} + \beta \Delta w_{ji}^{(q)}(t), \quad (11)$$

  where $0 < \beta < 1$ is a coefficient (often chosen to be 0.9), i.e. we add a *momentum or inertia* to smooth the drastic changes of successive learning directions.[41,86]
- A *single uniform and constant learning rate α is unable to suit the complex error-surface.*
  If α is too small, the learning will be very slow while if α is too large, the learning will 'overshoot' which again slows down the learning or even makes the learning diverge. Moreover, an appropriate α value at the beginning may not be so good later on. The main remedy for curing the problem is to adapt the learning rate as the learning progresses. Jacobs' DBD rule suggested two points for this adaption[102]:
  (1) each weight has its own learning rate,
  (2) α is increased when the derivative of a weight possesses the same sign for consecutive steps and α is decreased when the derivative of a weight changes sign.

  Several improved versions of the rule and other similar heuristics have been proposed by a number of authors (Refs. 216, 58, 33, 204, 228, 147, 189, 219, 38, 4 and 191). These techniques differ mainly in the following several issues:
  (1) How to use the increment of α, e.g. whether to let the increment be an additive factor or a multiplication factor, let the positive and negative increments be the same or different;
  (2) How to estimate the increment of α, for example, based on the minimum value of all the derivatives $\sigma'(y_j^{(q)})^4$ via a measure of the change of curvature,[58] by gradient correlation[191] or by analyzing the local extreme value of the derivative of error with respect to each weight[228];
  (3) How to coordinate the changes in α and β (Refs. 4, 216 and 147);
  (4) How to treat the 'overshoot' case,[216,147] e.g. unlearn the recently updated weight increments, either resetting the weights or choosing some weighted average of several past values that the weight took.
- *Premature saturation.*
  Due to the nonlinear sigmoid function given by Eq. (2), for a neuron $o_j^{(q)} = \sigma(y_j^{(q)})$ when $o_j^{(q)}$ is near its maximum or minimum (i.e. zero or one), its derivative $\sigma'(y_j^{(q)})$ is also near zero or one respectively. Since the derivative is a multiplication factor in Eq. (10), it follows that $\frac{\partial E_2}{\partial w_{ji}^{(q)}}$ and all the derivatives in the layers lower than this neuron will be near zero too. In these cases, the weights have almost no change and the corresponding error will stay almost constant (but maybe quite large) for a rather long period which can significantly slow down the learning process. This phenomenon is called *premature saturation*[135] or *a standstill state.*[248] The basic remedy is to let the slant parameter a in Eq. (2) be adjustable so that when the learning falls into *premature saturation* the slant parameter a is reduced appropriately. This goal can be fulfilled by different ways. In Ref. 248, a rule is proposed to first detect a learning standstill state and a is reduced from its normal value a_0 to a value such that the conditions for the learning standstill state are not satisfied, then make weight update once and switch a back to its normal value a_0. In Ref. 178, the parameter a is treated similarly to all the weights $w_{ij}^{(q)}$ and is updated in every learning step. While in Ref. 243, Eq. (1) is replaced by a version with normalized weights

$$y_j^q = \sum_{k=1}^{n_{q-1}} w_{jk}^{(q)} o_k^{(q-1)} \Big/ \sqrt{\sum_{k=1}^{n_{q-1}} (w_{jk}^{(q)})^2} \quad (12)$$

  and parameter a is also updated at the same time as updating all the weights $w_{ij}^{(q)}$. Here, not only the adjustments of a can evade the standstill state but also the normalized weights can avoid the cases that the large and small weights let the learning fall into the standstill state, thus the learning is more stable. In addition, the problems of a standstill state caused by the neurons in the

output layer can also be avoided by reconfigurating the error function such that the usual bell-type error shapes $\frac{\partial E_2}{\partial y_j^{(q)}}\sigma'(y_j^{(q)})$ for the neurons in the top layer can be changed into linear or sigmoid shapes (Refs. 17, 125 and 30).

- *Inappropriate initial weights.*
  It has been shown that the backpropagation learning technique is quite sensitive to the initial weights.[119] The inappropriate initial weights can either let the learning be stuck at local minima or considerably increase the learning time. In addition, the large or small weights will also let the initial learning fall into *premature saturation* which as stated above, makes the learning slow down considerably. Generally, the appropriate selection of good initial values is not an easy task. In Ref. 157, it is observed that in a one-hidden-layer multilayer net, each hidden unit tries to sample the desired mapping function specified by the training set. Thus, it is found that the learning can speed up significantly by picking the initial weights in such a way that all the hidden units are scattered uniformly in the input pattern space. In Ref. 135, the probability of *premature saturation* caused by the initial weights is calculated which is used as a guide for selecting the initial weights to avoid this situation.

In addition to the above heuristic techniques, other heuristics are also used to accelerate of backpropagation, e.g. in Ref. 77, a Hebbian unlearning term is attached to Eq. (8) to give $\Delta w_{ji}^{(q)} = -\alpha\frac{\partial E_2}{\partial w_{ji}^{(q)}} - \gamma o_j^{(q)} o_i^{(q-1)}$ with γ being a parameter $0 < \gamma < 1$; while in Refs. 87 and 138, a weight decay term is added, i.e. $\Delta w_{ji}^{(q)} = -\alpha\frac{\partial E_2}{\partial w_{ji}^{(q)}} - \gamma w_{ji}^{(q)}$. It was reported that both measures work.

Besides the above heuristic techniques for increasing convergence speed, a number of authors have also proposed to use other more advanced gradient techniques to replace the simplest gradient descent technique. Among them, the two major types are summarized as follows:

- *Conjugate gradient.*
  By this technique, the weight update is given by

$$\begin{aligned}\Delta\mathbf{w} &= \alpha\mathbf{d}^{\text{new}}\\ \mathbf{d}^{\text{new}} &= -\nabla E_2^{\text{new}} + \beta\mathbf{d}^{\text{old}},\end{aligned} \tag{13}$$

  where $\mathbf{w}$ consists of all the elements of the weights set $\{\mathbf{W}\}$ as components. The parameter β is selected such that $(\mathbf{d}^{\text{old}})^t H\mathbf{d}^{\text{new}} = 0$ for the Hessian matrix H. Such a β can be chosen by the *polakribiere* rule

$$\beta = \frac{(\nabla E_2^{\text{new}} - \nabla E_2^{\text{old}})^t \nabla E_2^{\text{new}}}{\|\nabla E_2^{\text{old}}\|^2}. \tag{14}$$

  The technique has advantages in both speed and convergence over steepest descent. Note that the computations involve only the derivatives $\frac{\partial E_2}{\partial w_{ji}^{(q)}}$ which can still be calculated by the backpropagation rule given in Eq. (9) and Eq. (10),[122,141] and no explicit knowledge of H is needed.
- *Newton's method.*
  The weights are updated by

$$\Delta\mathbf{w} = -\alpha H^{-1}\nabla E_2^{\text{new}}. \tag{15}$$

  The computation of this method is very expensive because of the need for inverting the Hessian H. This is usually not directly used in practice. Instead, the following *Quasi-Newton method* is used

$$\begin{aligned}\Delta\mathbf{w} &= -\alpha G^{\text{old}}\nabla E_2^{\text{old}},\\ G^{\text{new}} &= G^{\text{old}} + F(G^{\text{old}}, \Delta\mathbf{w}, \nabla E_2^{\text{new}} - \nabla E_2^{\text{old}}),\end{aligned} \tag{16}$$

  where G is the approximation of H^{-1} and the matrix F is a complicated function of its three arguments. Like the *conjugate gradient method*, the computations here also involve only the derivatives $\frac{\partial E_2}{\partial w_{ji}^{(q)}}$ which are obtained by the backpropagation rule give in Eqs. (9) and (10). It has been found[225,163] that the technique can increase the speed of gradient descent backpropagation by an order of magnitude.

  An alternative way is to approximate Newton's rule so that a matrix inversion is not required (Refs. 163, 179 and 21). One simple way is to use only the diagonal elements of the Hessian H by setting all the other elements to be zero, which simplifies Eq. (15) into

$$\Delta w_{ji}^{(q)} = -\alpha\frac{\partial E_2}{\partial w_{ji}^{(q)}}\Big/\frac{\partial E_2^2}{\partial w_{ji}^{(q)^2}} \tag{17}$$

  or its modified version (the c given below is a constant)

$$\Delta w_{ji}^{(q)} = -\alpha\frac{\partial E_2}{\partial w_{ji}^{(q)}}\Big/\left[\left|\frac{\partial E_2^2}{\partial w_{ji}^{(q)^2}}\right| + c^2\right]. \tag{18}$$

  However, this kind of *pseudo-Newton* rule usually cannot obtain comparable results to those obtained by *conjugate gradient* or *the quasi-newton method* and seems to give only a modest speed increase over the simple gradient descent method.

2.3. *Techniques for increasing generalization ability*

One goal of learning is to let the net *remember* as many as possible arbitrary mapping pairs $\{\mathbf{x}_p, \mathbf{d}_p, p = 1, \ldots N_p\}$[e]. It is not difficult to see intuitively that the larger the number of neurons, the larger the number of pairs the net can memorize. Given a training set D_{tr}, as long as the net has enough hidden units, the total error E_2 given in Eq. (7) can become very small or even zero after backpropagation learning has been implemented for a long enough period; i.e. a net with a large number of hidden units can memorize the training set very well. The other and even more important goal of learning is to make sure the net can *generalize* correctly to new mapping pairs after training on a training set of given mapping pairs; i.e. given a set of training data $D_{tr} = \{(\mathbf{x}_p, \mathbf{d}_p), p = 1, \ldots, N_p\}$ drawn from the desired function $\bar{F}(\mathbf{x})$ (maybe subject to noise), the mapping function $F(\mathbf{x})$ specified by the net after training can approximate $\bar{F}(\mathbf{x})$ well under a given measure not only on the samples from D_{tr} but also on the samples outside of D_{tr}. Such kind of ability is called the net's *generalization ability.*

Another drawback of a net trained by classical backpropagation is that it may have a poor generalization ability. Unlike the case of memorizing, a net having a large number of neurons may generalize very poorly if the number of samples in D_{tr} is not large enough. Roughly speaking, the generalization ability depends on the size $\#\Sigma$ of the set Σ given in Eq. (5) (i.e. the total number of functions implementable by the given net architecture), the size N_p of the training set D_{tr} and the size $\#\Sigma_{\bar{f}}$ of the subset $\Sigma_{\bar{f}} \subset \Sigma$ where each function $F(\mathbf{x}) \in \Sigma_{\bar{f}}$ can approximate $\bar{F}(\mathbf{x})$ well on the samples from and outside of D_{tr}, i.e.

$$\Sigma_{\bar{f}} = \{F(\mathbf{x}) | F(\mathbf{x}) \in \Sigma, \quad \text{for } \mathbf{x} \in D, \\ m_e(F(\mathbf{x}), \bar{F}(\mathbf{x}) \leq \varepsilon_t\}, \tag{19}$$

with $D = R^m$ or $D \subset R^m$ being the domain of $\mathbf{x}$ which may be much larger than the subset $\{x_i, i = 1, \ldots N_p\}$ occupied by the training set D_{tr}, where $m_e(\mathbf{a}, \mathbf{b}) \geq 0$ is a given error measure (e.g. $m_e(\mathbf{a}, \mathbf{b}) = ||\mathbf{a} - \mathbf{b}||^2$) and ε is a prespecified error bound. The larger the number of neurons in a net, the larger the number of free weight parameters in the weight set $\mathbf{W}$ and thus the larger $\#\Sigma$. In this case, a large enough N_p is required; otherwise, a poor generalization will be obtained since there are a vast number of functions in Σ which are consistent with the training set D_{tr} but not all are in $\Sigma_{\bar{f}}$ and hence the function specified after training is unlikely to be in $\Sigma_{\bar{f}}$. However, on the other hand, if the number of neurons in the net and thus the number of free weight parameters is not big enough, there may be no function in Σ which can approximate $\bar{F}(\mathbf{x})$ well and so $\Sigma_{\bar{f}}$ may be empty. i.e. the net obtained after training might even have a poor memorizing ability. An illuminating analogy can be drawn between the learning of a net and the fitting of a curve by a parametric model. When the model has a lot of parameters, one encounters the *overfitting* problem which prevents smooth interpolations and reasonable extrapolations; on the other hand, when the model has too few free parameters, one meets the *underfitting* problem which again gives a poor result.

In recent years, a number of methods have been made to improve the generalization ability of a net trained by classical backpropagation. The key ideas behind these methods are to reduce the number of neurons or the number of free parameters in order to cut down the size $\#\Sigma$ while at the same time not to overly reduce the net's memorizing ability. These methods can be summarized into two groups. The *first* group consists of methods which impose certain constraints on the values taken by the weights in a network architecture without modifying the net's hardware. The *second* group contains the methods which dynamically adjust the number of neurons (usually in the hidden layer) and thus modify the net's hardware. In the sequel, we survey the major methods in each of the two groups.

In the first group, the proposed major methods are the following types:

- *Task-dependent constraints.*
  For the specific applications, there is sometimes some task-dependent prior information which can be used to impose constraints on the weights of the net, e.g. equality constraints can be used to implement the so called *weight sharing* technique[183,130] (i.e. let several weights be controlled by a single parameter) which has been successfully used in robust handwritten zip code recognition;[131] others like symmetry, order, geometrical and topological constraints can also be used in some problems.[206,84]
- *Distributed bottleneck.*
  Suppose that the kth layer has B neurons and the

[e] When $\mathbf{d}_p$ is a binary vector (i.e. its component takes binary values), N_p is called the net's capacity.

$(k-1)$th layer has A and let W_{BA} denote the weight matrix that connects the $(k-1)$ layer to the kth layer; the idea of this method is to impose a constraint on the rank of W_{BA} such that it can be written as $W_{BA} = W_{BQ}W_{QA}$ and $B > Q < A$. This is equivalent to creating a "virtual bottleneck layer" of Q linear neurons between the kth and $(k-1)$th layers. The method does not explicitly cut out some neurons but causes the neurons to be as similar as possible, i.e. causing them to become redundant. The goal is realized by simultaneously compressing the weight vectors of W_{BA} into a low dimensional space and clustering them within the low dimensional space.[124]

- *The minimum network.*
  The idea is to force the "useless" weights to become zero and thus be implicitly deleted to obtain a minimum network. The idea is implemented through adding a term in the error function that penalizes big nets with many weights (Refs. 34, 35, 87, 80, 153 and 101), i.e. to let Eq. (7) be replaced by E_2+cost. Where the simplest cost term is cost$= b\sum_{w_{ij}\in \mathbf{W}} w_{ij}^2$ which is sometimes called ridge regression in statistical theory. This extra cost term is equivalent to put a weight decay term in the gradient formula Eq. (8), i.e.

$$\Delta w_{ji}^{(q)} = -\alpha \frac{\partial E_2}{\partial w_{ji}^{(q)}} + b w_{ji}^{(q)}, \qquad (20)$$

  where b is an adjustable constant that usually starts from zero and then gradually increases to some value in order to gradually introduce the decaying force which tends to prevent a net from using table lookup and forces it to discover regularities in the training set.[87] In Ref. 34, a more complicated cost is used in a one-hidden-layer net, given by

$$E_2 + a \sum_{\text{all hidden units}} \frac{o_i^2}{1+o_i^2} + b \sum_{\text{all } w_{ij}\in \mathbf{W}} \frac{w_{ij}^2}{1+w_{ij}^2}, \qquad (21)$$

  which also tends to force the used hidden units to be as small as possible. In addition, it tries to slow down the decay of large value weights to avoid washing away what was already learned earlier. Another extended version of the cost function is given as follows[227]

$$E_2 + b \sum_{\text{all } w_{ij}\in \mathbf{W}} \frac{w_{ij}^2/c^2}{1+w_{ij}^2/c^2}, \qquad (22)$$

  where c is a constant and it reduces to the usual ridge regression case Eq. (20) when $c \to \infty$. It is interpreted by Ref. 227 to be related to the information theoretic minimum description length (MDL) criterion.
- *Other global penalizing term.*
  Penalizing terms other than the above *minimum strategy* may also be used for improving the generalization ability, e.g. in Ref. 54, an additional cost term $\sum_i (\frac{\partial E_2}{\partial x_i})^2$ is also minimized together with E_2 where x_i is a component of $\mathbf{x} = [x_1, \ldots x_n]^t$. The motivation is based on their observation: after training, if all gradients at the output layer are zero, then the gradients at the input $\mathbf{x}$ should also be zero. However, in practice, this is usually never true. Thus, in order to let the output be less sensitive to amplitude variations of the input so that a smoother approximation is obtained, they thought that it would be desirable to force the input gradients to be zero even if the gradients of the preceeding layers are nonzero.

Like the second of the above methods, the purpose of the methods in the second group is also to create a bottleneck layer (i.e. the hidden layer for the commonly used one-hidden-layer net) in the network's architecture. The key difference here is that the bottleneck is not distributed but is obtained by dynamically pruning or deactivating some neurons in the hardware. The main techniques of this group can be classified into the following three types:

- *Selecting an appropriate architecture according to some error criteria which take generalization ability into consideration.*
  The basic idea is to select the best, under a given criterion, of a number of architectures trained by backpropagation separately. The implementation of the idea is usually to start from an architecture with a given small number of neurons in the bottleneck layer and then gradually to increase the number of neurons to produce the next architecture until the criterion value is below a prespecified threshold or reaches a minimum and tends to rise again. There are several error criteria proposed for this purpose. Here, we list some of them. One includes simultaneously the error terms $E_{\text{approx.}} + E_{\text{gener.}}$[13,136] $E_{\text{approx.}}$ is just E_2 given in Eq. (7) or the classification error when the net is used for pattern recognition tasks. E_{gener} describes the generalization error, e.g. in Ref. 136, it is specified by $b\sqrt{(d_{VC}\beta/N_p)\ln(N_p/d_{VC})}$ with b being a weighting factor and d_{VC} being the so-called Vapnik-Chervonenkis dimension. Some similar but different criteria are given in Ref. 13.

The second one is the prediction probability given by Ref. 215. It gives the probability that a net, trained on a training set, can correctly predict an example which is independent of the training set. The probability is highly correlated with the generalization ability of the net, as measured outside of the given training set. The third one is the hypothesis testing criterion given by Ref. 234. The idea is to test whether a trained net represents exactly a given mapping function subject to inherent noise or whether there are some nonlinear structures in the mapping neglected by the net, based on the methods that determine whether or not there exists some advantage to be gained by adding hidden units to the given net. The fourth is the information theoretic measure given in Ref. 221.

- *Heuristic dynamical adjustment of the number of neurons in the bottleneck layer.*
  There are several ways. The first one is to start from a large number of neurons, and then decrementally reduce the number by some heuristics, e.g. in Ref. 78, a "badness factor" describing the backpropagation error for each hidden neuron is used to detect the worst hidden neuron to prune off; while in Ref. 110, the redundant neurons were pruned away by a heuristic sensitivity measure. The second way is to start from a small number of neurons and incrementally increase the number of neurons heuristically until some predefined condition is satisified and then start to prune neurons one by one until the condition no longer holds e.g. in Ref. 85, the error E_2 given in Eq. (7) is checked after every 100 learning updates, if it does not decrease by more than one percent of its previous values, a new hidden neuron is added. Otherwise, another 100 weight updates are made. Once the net reaches $E_2 = 0$, one starts to remove a neuron and the net is trained again, if still $E_2 = 0$ holds, then remove another until it no longer holds. The third way is to remove from a trained net the neurons which are regarded as redundant by some heuristic judgements, e.g. in Ref. 203, to prune away:

  (1) neurons which either have approximately constant output across the training set or have outputs across the training set which mimic the outputs of other units.
  (2) neurons which are independent of the other units in the layer but given information that is not required at the next layer.

  The fourth way is to estimate the expected number of neurons in the bottleneck layer by some heuristic method, e.g. in Ref. 126, subspace projection with least square approximation is used to select the best subset of hidden neurons. Another example for estimating the expected number of neurons is a net with binary input given by Ref. 76.

It is not difficult to see that the methods of the second group above can also help to speed up the training process of a net because of the reduction of the number of hidden units. Moreover, besides the two groups of methods, there are also some other ways for improving the net's generalization ability, e.g. Ref. 19 proposed an algorithm which constructs hidden units using examples and queries.

2.4. *Criteria of different types*

The core of the backpropagation technique described in Sec. 2.1 is to use the chain rule on the compound function given in Eq. (3) to obtain derivatives of E_2 given in Eq. (7) with respect to weights of the nonoutput layer. Without changing this core, we can still obtain a number of other variants of backpropagation by modifying some other factors, such as by using different criteria and activation functions etc. This subsection with survey the variants resulted from using different criteria and the next subsection will review some other variants.

According to their characteristics, the presently used criteria can be roughly grouped into the following five types.

1. *Minkowski-r error (or L_r norm).*
   The criterion is given by

$$E_r = \sum_{p=1}^{P} \varepsilon_p^r$$
$$\varepsilon_p^r = \sum_{i=1}^{m} |d_{pi} - f_i(\mathbf{x}_p)|^r , \tag{23}$$

   where

$$\mathbf{d}_p = [d_{p1}, \cdots, d_{pm}]^t,$$
$$F(\mathbf{x}_p) = [f_1(\mathbf{x}_p), \cdots, f_m(\mathbf{x}_p)]^t$$

   and $1 \leq r < \infty$. Obviously, the criterion is a direct extension of the least square error since it includes Eq. (7) as the special case $r = 2$. As pointed out in Refs. 31, 81, 151 and 152, the least square criterion is suitable only when the probability distribution of error noise $\mathbf{e}_n = \mathbf{d} - F(\mathbf{x})$ is Gaussian. However, when the

distribution is Laplace (e.g. the exponential distribution), it is best to use Eq. (23) with $r = 1$. When the distribution is uniform, it is best to use Eq. (23) with $r = +\infty$. The smaller $r < 2$ is generally suitable for suppressing noises (may vary with the problem and the nature of noise); the larger $r > 2$ tends to weight large deviations and increase the sensitivity to the geometry of the desired patterns.

This often used criterion has some important theoretical interpretations. The minimization of E_r given in Eq. (23) is equivalent to letting $F(\mathbf{x})$ to be the maximum likelihood estimation of the desired function $\mathbf{d}(\mathbf{x})$ in three special case (Refs. 151, 152, 81, 13 and 31):

(i) for $r = 2$, when $\mathbf{e}_n$ obeys a Gaussian distribution,
(ii) for $r = 1$ when e_n obeys a Laplace (e.g. exponential) distribution,
(iii) for $r = +\infty, e_n$ obeys a uniform distribution.

Furthermore, for the case that $r = 2$ and the net is used as a classifier (i.e. for the desired $\mathbf{d}(\mathbf{x})$, one has $d_i(\mathbf{x}) = 1$, if $\mathbf{x}$ belong to class C_i, otherwise $d_i(\mathbf{x}) = 0$). It can be shown (Refs. 55, 71, 184, 223, 109 and 200) that minimization of E_2 is equivalent to choosing $f_i(\mathbf{x})$ given in Eq. (3) to approximate $p(C_i/\mathbf{x})$ in the least square sense with $p(C_i/\mathbf{x})$ being the posterior probability of class C_i; i.e., $\min E(\sum_{i=1}^{m}[d_i(\mathbf{x}) - f_i(\mathbf{x})]^2)$ is equivalent to $\min E(\sum_{i=1}^{m}[p(C_i/\mathbf{x}) - f_i(\mathbf{x})]^2)$. Thus, classifying $\mathbf{x}$ according to $c = \arg\max_i f_i(\mathbf{x})$ by the net can be regarded as a kind of approximation of classifying $\mathbf{x}$ according to Bayesian decision rule $c = \arg\max_i p(C_i/\mathbf{x})$.

Presently, there are two ways to decide the value of r for using E_r. One is to select r externally according to the analysis results of the problem and the nature of noise. The other way is to modify r also by gradient descent method through solving $\frac{\partial E_r}{\partial r}$ which is easy to obtain and does not need the use of the chain rule. However, as indicated in Ref. 81, it is important that the update of r should be several times slower than the updates of weights. The details for the derivatives of E_r are given in Ref. 81, and for the special case of $r = 1$ and $r = +\infty$ are given in Ref. 31.

2. *Entropy-like criteria.*
There are several variants. The most popular is given as follows:

$$\begin{aligned} J_e &= \sum_{p=1}^{P} \eta_p \\ \eta_p &= -\sum_{i=1}^{m}[d_{pi}(\mathbf{x}) \ln f_i(\mathbf{x}_p) \\ &\quad + (1 - d_{pi}(\mathbf{x})) \ln(1 - f_i(\mathbf{x}_p))] . \end{aligned} \tag{24}$$

The criterion has one interesting feature. When the activation function of the neurons in the output layer is sigmoid and given by Eq. (2), the derivatives for the output layer is given by

$$\begin{aligned} \frac{\partial J_e}{\partial y_j^{(L)}} &= \sum_{p=1}^{P} \frac{\partial \eta_p}{\partial y_j^{(L)}} \\ \frac{\partial \eta_p}{\partial y_j^{(L)}} &= \sum_{i=1}^{m}[d_{pi}(\mathbf{x}) - f_i(\mathbf{x}_p)] , \end{aligned} \tag{25}$$

instead of that for E_2, given by

$$\begin{aligned} \frac{\partial E_2}{\partial y_j^{(L)}} &= \sum_{p=1}^{P} \frac{\partial \epsilon_p^2}{\partial y_j^{(L)}} , \\ \frac{\partial \epsilon_p^2}{\partial y_j^{(L)}} &= 2\sum_{i=1}^{m}[d_{pi}(\mathbf{x}) \\ &\quad - f_i(\mathbf{x}_p)] f_i(\mathbf{x}_p)[1 - f_i(\mathbf{x}_p)] . \end{aligned} \tag{26}$$

That is, the factor $f_i(\mathbf{x}_p)[1 - f_i(\mathbf{x}_p)]$ has been removed in Eq. (25). As mentioned in Sec. 2.2, the factor is responsible for premature saturation problem which can slow down the learning speed. Thus, the removal of the factor in the output layer by criterion Eq. (24) can speed up learning and improve convergence. This feature was realized earlier by Refs. 17 and 87 and later experimentally verified by Refs. 125, 82 and 89.

The criterion also has several theoretically justified interpretations. First, when $f_i(\mathbf{x})$ is considered as the estimate of the posterior probability $\hat{p}(C_i/\mathbf{x}, W)$ given the weight set W, the minimization of Eq. (24) is equivalent to maximize the following likelihood function

$$\begin{aligned} \mathrm{Max}_W\ L = \mathrm{Max}_W \prod_{i=1}^{m} \hat{p}(C_i/\mathbf{x}, W)^{d_i(\mathbf{x})} \\ \times [1 - \hat{p}(C_i/\mathbf{x}, W)]^{(1-d_i(\mathbf{x}))} . \end{aligned} \tag{27}$$

Second, if $d_i(\mathbf{x})$ is also regarded as a probability (taking values in [0,1]), Eq. (24) is also interpreted as an *entropy* by Ref. [17] or a *cross entropy* by Ref. 87. Third, by taking expectation of η_p, one can also show[244] that the maximization of $E(\eta)$ is equivalent to estimating the posterior probability $p(C_i/\mathbf{x})$ under an extended Kullback-Leibler information measure. In Ref. 57, one other entropy-like measure is suggested by modifying η_p in Eq. (24) into

$$\eta_p = -\sum_{i=1}^{m} \ln[f_i(\mathbf{x}_p) - (1 - d_{pi}(\mathbf{x}))]^2 . \quad (28)$$

It is shown that the criterion can also improve convergence.

3. *Heuristic criteria for classification purpose.* When a net is used as a classifier, one can reach a right decision as long as $f_i(\mathbf{x}_p) = \max_k f_k(\mathbf{x}_p)$ and $f_i(\mathbf{x}_p) \geq t_h$ (a threshold, often 0.5) even when the least square error $\sum_{i=1}^{m}[d_{pi}(\mathbf{x}) - f_i(\mathbf{x}_p)]^2$ can be quite large. This gives us a hint that the large values of $d_{pi}(\mathbf{x}) - f_i(\mathbf{x}_p)$ are more influential for the right classification than the smaller ones. Based on this observation, a modification of the least square error E_2 is given as follows (Refs. 252, 94 and 4)

$$E_2' = \sum_{i=1}^{m} \max\{[d_{pi}(\mathbf{x}) - f_i(\mathbf{x}_p)]^2, c\}, \quad (29)$$

with c being a positive constant parameter. Usually, it is not easy to select an appropriate value for c. Thus, in practice, c is initially given a rather large value and gradually reduced to a small value according to a certain schedule similar to that of simulated annealing.[113] It has been observed that such a criterion cannot only improve the performance of classification but also speed up convergence and avoid local minima.[252,4] A different but somewhat similar criterion is proposed by Ref. 141 which is given by

$$E_2' = \sum_{i=1}^{m} \times \begin{cases} c[d_{pi}(\mathbf{x}) - f_i(\mathbf{x}_p)]^2 & \text{if } d_{pi}(\mathbf{x}) f_i(\mathbf{x}_p) > 0 \\ [d_{pi}(\mathbf{x}) - f_i(\mathbf{x}_p)]^2 & \text{otherwise}, \end{cases} \quad (30)$$

with c gradually increasing from 0 to 1.

Reference 79 proposed a more sophisticated criterion for classification purpose. The criterion focus most heavily on the reduction of misclassification rate through maximizing

$$J_c = \sum_{p=1}^{P} \left(\sum_{i \neq r}^{m} \frac{\alpha}{1 + \exp(-\beta \Delta_i + \xi)} \right), \quad (31)$$

where $\alpha, \beta \geq 0$ are parameters, r is the index of the right class (i.e. $d_r = 1, d_i = 0$ for all $i \neq r$), and $\Delta_i = f_i(\mathbf{x}_p) - f_r(\mathbf{x}_p)$. It has been shown experimentally that the criterion can improve classification performance significantly. Moreover, it is also proved in Ref. 11 that the criterion is equivalent to the Bayes rule for the special case that the input x is one dimensional.

4. *Heuristic criteria for speeding up.* The key idea of these heuristics is to modify the shape of $\frac{\partial \epsilon_p^2}{\partial y_j^{(L)}}$ given in Eq. (26) so that the negative influence of the factor $f_i(\mathbf{x}_p)[1 - f_i(\mathbf{x}_p)]$ can be removed. As shown above, the entropy criterion Eq. (24) can reach this goal. However, in addition to this criterion, there are also several other heuristic ways; in Ref. 60, the goal is achieved by a modified least square criterion

$$\epsilon_p^2 = \sum_{i=1}^{m} [\sigma^{-1}(d_{pi}(\mathbf{x})) - \sigma^{-1}(f_i(\mathbf{x}_p))]^2, \quad (32)$$

where $\sigma^{-1}(.)$ is the inverse of the activation function $\sigma(.)$ of the output neurons. In Ref. 108, it is shown by some experiments that the following modification can also speed up convergence considerably

$$\epsilon_p^2 = \sum_{i=1}^{m} [d_{pi}(\mathbf{x}) - f_i(\mathbf{x}_p)]^2 / [1 - f_i^2(\mathbf{x}_p)]. \quad (33)$$

Furthermore, it has been noticed by Refs. 30 and 58 that for the least square error E_2, the derivatives $\delta(e_{pi}) = \frac{\partial \epsilon_p^2}{\partial y_j^{(L)}}$ has a Bell or Bump-like shape with error $e_{pi} = d_{pi}(\mathbf{x}) - f_i(\mathbf{x}_p)$, i.e. $\delta(e_{pi})$ has values only within a range in which $|e_{pi}|$ is smaller than certain values. Because $\delta(e_{pi})$ becomes very small or zero when $|e_{pi}|$ is large, the learning increment can be very small even when errors are large. The remedying policy is to redesign a criterion such that $\delta(e_{pi})$ has a linear or sigmoid like shape. Two examples of such criteria are given by Refs. 30 and 58 respectively. In addition, it is not difficult to see that the criteria given in

Eqs. (32) and (33) are actually also consistent with this policy.

5. *Criteria for improving generalization.*
   There are a number of such criteria which we have already summarized in Sec. 2.3.

As described above, each type of criteria has its own characteristics. In practice, the selection of an appropriate one is usually problem-dependent. Roughly speaking, when the net is used for a general function approximating task, the Minkowski r-error with $1 \leq r \leq 2$ is more appropriate. In particular, that with $r = 2$ (i.e. the least square error E_2) is the most popular used one. When the net is used as a classifier, then the criteria of the 2nd and 3rd types as well as the Minkowski r-error with larger r are better. We believe that investigations are still needed for further comparison and verification of these different criteria.

Before closing this subsection, we would also like to mention that there are also some other types of criteria that have been suggested in recent years, e.g. in Ref. 127, it is reported that the minimization of a weighted mixture of the mean and the variance of the least square error e_{pi}^2 may give a more robust performance. However, due to the limited space, we omit here those miscellaneous issues.

2.5. *Some other variants*

There are also a number of other types of variants which are obtained by keeping the key feature of backpropagation while varying some other issues, e.g.

(1) let the sigmoid function of the output neurons be replaced simply by the linear function (i.e. let $\sigma(y) = y$) such that the net is more suitable for approximating arbitrary functions (Refs. 91, 92, 93, 211, 123 and 235);
(2) to constrain the weights in a net to take only a finite set of discrete values (i.e. three values in Ref. 200 and finite precision integers in Ref. 90) or bounded values[212] or normalized values[243] in order to facilitate hardware implementations and improve performance;
(3) to extend the real valued inputs, outputs and weights into complex valued domains in order to tackle some signal processing problem[112];
(4) to replace the simple summation-sigmoid neurons of Eqs. (1) and (2) by more complicated neurons too get some desired features.

The above first three types are quite straightforward. Thus, in the sequel, we review in detail several examples of the last type:

- *Using other activation functions.*
  The simplest version is to use Gaussian function $g(y) = \exp(-y^2/a^2)$ to replace the sigmoid function $\sigma(y)$ used in Eq. (2). One such example can be found in Ref. 97 where $g(y)$ is used in the hidden layer of an one-hidden layer net with the neurons in the output layer being simply linear. The net is still trained by the chain rule backpropagation method and experiments have shown that it is more suitable for representing EMG(electromyogram) signal. One other example is given in Ref. 243, it has been shown that only one of the following normalized Gaussian neurons

$$o_j^{(q)} = g(y_j^{(q)})$$
$$y_j^{(q)} = \sum_{k=1}^{n_{(q-1)}} w_{jk}^{(q)} o_k^{(q-1)} \Big/ \sqrt{\sum_{k=1}^{n_{(q-1)}} [w_{jk}^{(q)}]^2}\,, \tag{34}$$

  can beat the XOR problem effectively and it has been further shown that a net of one-hidden layer, with hidden neuron given by Eq. (34) and only one linear output neuron, can solve the hard classification problem given in Ref. 96 with considerably less neurons and fast training speed. It has been also suggested in Ref. 243 that the hybrid use of sigmoid neuron Eqs. (1) and (2) and Gaussian neuron Eq. (34) in the same hidden layer will be more effective in reducing both the number of neurons and the training times. There is also another type of Gaussian neuron which has been widely used in the literature recently. The neuron is simply given by $o_j^q = g(||w_{jk}^{(q)} - \mathbf{x}||^2)$ and just used in the hidden layer of a one-hidden-layer net called *Radial Basis Function* net. Strictly speaking, this kind of net is trained no longer by the chain-rule-based-backpropagation technique, thus, we will postpone the survey of such nets until later in Sec. 4.

  Besides Gaussian activation functions, some other types of functions have also been tried; e.g. in Ref. 132, sine function $s(y) = \sin y$ is used to replace $\sigma(y)$ in Eqs. (1) Eq. (2) to give an accurate computation of forward kinematic solution of a robot arm.
- *Using high order neurons*
  I.e. to replace Eq. (1) by a polynomial of order r,

e.g. when $r = 2$,

$$y_j^q = w_{j0} + \sum_{k=1}^{n_{q-1}} w_{jk}^{(q)} o_k^{(q-1)} + \sum_{i=1}^{n_{q-1}} \sum_{k=1}^{n_{q-1}} w_{jik}^{(q)} o_i^{(q-1)} o_k^{(q-1)}, \quad (35)$$

with w_{jik} being 2nd order connection weights. The net with high order neurons can treat some complex problems with a few number of neurons. In principle, the chain rule based backpropagation can be still used to train such a net. The disadvantage of the net is that it requires a large number of high order weights due to combinatorial explosions. Usually, some *a priori* knowledge about inputs should be preset in the net.[70] A variant of high order neuron is recently proposed by Ref. 199 which is given by

$$y_j^q = \prod_{i=1}^{r} \left(\sum_{k=1}^{n_{q-1}} w_{kij}^{(q)} o_k^{(q-1)} \right). \quad (36)$$

This variant can solve the combinatorial explosion problem of high-order weights and convergence improvement is reported too.

- *Using neurons in clusters.*
  Reference 52 suggested replacing each hidden neurons in a one-hidden-layer- net by a cluster of neurons which are constrained to behave in a coordinated way. The net is trained by backpropagation in such a way that each time only one neuron in each cluster is updated in the usual backpropagation sense. This update causes a change in the centroid of the cluster and all the neurons in this cluster are updated in a coordinated way according to the centroid change. Such a net is believed to have fault tolerance ability and can speed up training. However, it seems that some further studies (especially experiments) are needed on this sort of variant.
- *B-spline receptive field function.*
  The variant is proposed in Ref. 128. The idea to replace Eq. (1) by

$$y_j^q = \sum_{k=1}^{n_{q-1}} \sum_{r} w_{jkr}^{(q)} B_{nr}(o_k^{(q-1)}), \quad (37)$$

where $B_{nr}(y)$ is one-dimensional spline of order n. We see that a weight $w_{jr}^{(q)}$ in Eq. (1) is split into a group of weights $w_{jkr}^{(q)}$ for suming several transformed values of the output $o_k^{(q-1)}$ by the spline functions $B_{nr}(y)$'s, called B-spline receptive field functions. These split weights are updated again by backpropagation based gradient descent rule.

3. Beyond Backpropagation: Other Learning Techniques for Multilayer Perceptron

Although backpropagation was and still is the most widely used technique for training multilayer perceptron, its monopoly role is now being shared by a number of other learning methods which have been increasingly used for training the architecture of the multilayer perceptrons given in Sec. 2.1 especially in the recent two years.

These alternative methods may be roughly classified into two groups according to their approaches on weight updating. For the methods of the first group, like the backpropagation method, the amount by which each weight should to be updated is calculated based on the current final error value E_2 and on the transfer function between the weight and the final E_2; therefore prior knowledge of the networks' internal transfer characteristics is crucial to these methods. By contrast, for the methods of the second group, the amount of each weight update is not calculated from the present E_2 value but is obtained by some random or heuristic variations or perturbations. No knowledge of networks' transfer function is needed and the weight update can be decided just by measuring the output E_2 by regarding the net as a black box.

For convenience, we reverse the order and first consider the methods of the second group. This group consists of three types of methods:

(1) derivative estimation by perturbation, e.g. MRIII[7] and model free learning[50];
(2) direct weight update by perturbation, e.g. local variation,[165] random optimization[9] and simulated annealing[49];
(3) genetic algorithms.

These methods have two common advantages. The first one is that they are more suitable for implementation by analog (or partially analog) circuits which can provide several advantages over digit circuits in terms of cost, packing density, power usage and in some cases, speed, since there is no need to know *a priori* accurate the networks' architecture and the transfer characteristics of the computing devices, which are expensive to obtain accurately for analog devices; thus the analog devices are not

suitable for learning methods with features of weight updating similar to backpropagation. The second advantage is that it is also suitable for those nets consisting of neurons with nondifferentiable activation function which cannot be handled by the gradient based techniques like backpropagation. Furthermore, the method of the last two types have also the advantage that it can avoid local minima for getting a global optimal solution. The main disadvantage of this group is that most methods are slow in learning speed except for some special cases. In the sequel, the details of these methods are further reviewed:

- *Derivative estimation by perturbation.*
  Instead of estimating $\Delta W = -\alpha \frac{\partial E_2}{\partial W}$ in the top down way through the chain rule based backpropagation in a net, the key idea of *estimating derivatives by perturbation* is to first inject certain perturbations into the net and to forward calculate the change δE_2 caused by the perturbations and then to use this δE_2 and the perturbations to estimate ΔW or $\frac{\partial E_2}{\partial W}$. One representative of this type of techniques is MRIII.[7,240] By this method, each time one neuron (say that one given in Eq. (1)) is chosen randomly or circularly and a small perturbation δ is injected on $y_j^{(q)}$. The perturbed $y_j^{(q)} + \delta$ is passed up to the upper layers which causes a change in the final least square error E_2, denoted by $\epsilon_j^{(q)}$. Then $\epsilon_j^{(q)}/\delta$ is used as an approximation of $\frac{\partial E_2}{\partial y_j^{(q)}}$. As a result, one can obtain the following estimates

$$\delta w_{jk}^q = -\alpha \frac{\partial E_2}{\partial y_j^q} \frac{\partial y_j^q}{\partial w_{jk}^q} \approx -\alpha \frac{\epsilon_j^{(q)}}{\delta} o_k^{(q-1)}, \quad k = 1, \ldots, n_{q-1}, \tag{38}$$

  it is not difficult to see from Eq. (9) that the above method is just a kind of approximation of backpropagation. The other representative is the *model free distributed learning* technique given by Ref. 50 adapted from the "M.I.T. rule" in the field of adaptive control. The method has two points of differences from the above one. First, each of all the weights in a net is injected by a perturbation signal and these signal are mutually uncorrelated with zero means. Second, each time all the derivatives $\frac{\partial E_2}{\partial w_{jk}^q}$ are simultaneously estimated from these perturbation signals and the change δE_2 caused by these signals. The detailed theoretical analysis of the method is also available in Ref. 50.

- *Directly weight update by perturbation.*
  The key idea is to perturb each weight and to observe whether the change ΔE_2 caused by the perturbation is acceptable. If it is acceptable, the perturbation is granted; otherwise the perturbation is discarded. The idea is somewhat similar to the above technique, however, the main difference is not to estimate derivatives $\frac{\partial E_2}{\partial w_{jk}^q}$ but just to update w_{jk}^q by the perturbation if ΔE_2 is acceptable. Here we introduce three algorithms of this type of technique.

  The first is given by Ref. 165. By this algorithm, at each time t, a weight w_{jk}^q is chosen randomly or circularly and a positive variation $\delta > 0$ is made to produce $w_{jk}^{q'} = w_{jk}^q(t) + \delta$ which causes a change ΔE_2. If $\Delta E_2 < 0$, then $w_{jk}^q(t+1) = w_{jk}^{q'}$; otherwise make a negative variation $w_{jk}^{q'} = w_{jk}^q(t) - \delta$ and similarly let $w_{jk}^q(t+1) = w_{jk}^{q'}$ if $\Delta E_2 < 0$, otherwise go to next weight. The algorithm is iterated with time until E_2 is below a preset limit or a minimum. It is interesting to note that whenever a local variation is attempted on a given weight, the entire forward propagation for patterns need not be carried out. Instead, only the variables along a path affected by the variation should be re-evaluated. This requires some extra variables such as activation and outputs of neurons to be stored for each pattern in the pattern set. However, it can bring significant reduction in computation for every pattern presentation.

  The second algorithm is proposed by Ref. 9. The algorithm perturb all the weights in the net at each step instead of just one weight each step. By the algorithm, at each step, a perturbing matrix $\Delta(t)$ is generated from a Gaussian distribution with mean matrix $B(t)$ to perturb the weight matrix $W(t)$ in such a way that:

  Let $W' = W(t) + \Delta(t)$ and see whether the correspondent $\Delta E_2 < 0$; if yes, let $W(t+1) = W'$ and $B(t+1) = 0.4\Delta(t) + 0.2B(t)$; otherwise let $W' = W(t) - \Delta(t)$ and see if $\Delta E_2 < 0$; if yes, let $W(t+1) = W'$ and $B(t+1) = B(t) - 0.4\Delta(t)$; if again not, $W(t+1) = W(t)$ and $B(t+1) = 0.5B(t)$.

  The initial $W(0)$ is set randomly and $B(0)$ is zero matrix. In fact, the algorithm is a direct application of the modified random optimization method given by Ref. 207.
  One may already notice that the above two algorithms have the common point that a perturba-

tion is accepted only when $\Delta E_2 < 0$. This condition can be relaxed by using the simulated annealing technique[113] for escaping local minima. This idea leads to the third algorithm: let the acceptance of a perturbation be made by the probability $\exp(-\Delta E_2/T)$, T is a temperature parameter which starts from an initial value and gradually reduces to zero as learning goes. An example of such algorithm is given in Ref. 49.

- *Using genetic algorithm.*
  The idea is to apply Holland's genetic algorithm[73] for optimizing weights or even architecture of a neural net. In Refs. 236 and 149, the problem of training the weights of a net with fixed architecture is studied. The whole set W of weights in the net is encoded into a binary string (called a chromosome) which has an associated fitness value to describe the string's goodness. The fitness can be just the negative of the error criterion value, e.g. $-E_2$ when using the least square error. Initially, a population of strings is generated randomly and then *genetic operators* are used on the population to produce new strings. Usually, there are two kinds of such operators. One is "crossover" which exchanges part of one string with part of another to produce two new strings. The other is "mutation" which randomly alters bits of a string to give a new string. The "mutation" operation is usually used at a considerably lower frequency than the "crossover" operation. During these genetic operations, some strings of the population will die when their fitness values are weaker than others. After enough many generations, the final solution can be obtained by choosing the best fit string (or even by randomly choosing one).

  In Refs. 237 and 146, a genetic algorithm is also used to prune the networks' architecture by encoding the networks' connectivity into a string. In this case, the evaluation of the fitness of a string is very expensive: for each string (i.e. a network architecture), backpropagation is used to train it and then its classification rate is calculated on the testing samples. To reduce training times, Ref. 237 suggested training a net with a reduced number of training circle. Very recently, there is also an attempt to use the genetic algorithm to simultaneously optimize the networks architecture and weights.[120] The key points of using the genetic algorithm are to have an appropriate coding and a good design of "crossover" and "mutation" operators. Usually, the resulted code is very lengthy and large costs in both speed and storage are needed for a net of large size. Many efforts are still needed to make the genetic algorithm practical for a net of large size. Perhaps, combining the genetic algorithm with other learning techniques may give some chances for improvements, e.g. in Ref. 149, the combination of the genetic algorithm and backpropagation can outperform either method alone.

Now, we continue to consider the methods of the first group. As pointed out previously, for these methods the amount by which each weight should be updated is calculated from the final error E_2 and the net's internal transfer function. This group can still be further divided into two subgroups. The methods of the first subgroup still need to calculate derivatives by using the chain rule or directly using Eqs. (9) and (10) although the update $\Delta w_{ji}^{(q)}$ is not made in a gradient descent way. While the methods of the second subgroup no longer use the chain rule at all and the weight updates are made almost in a layer-by-layer decoupled manner. In the sequel, we review in detail the main methods and the characteristics of both the subgroups. These methods are grouped into five types, among which the first two belong to the first subgroup and the other three to the second subgroup.

1. *Recursive least square estimate and Extended Kalman filter.*
   Let us rewrite Eq. (7) as

$$\epsilon_k = \mathbf{d}_k - F(\mathbf{x}_k, \mathbf{w}), \tag{39}$$

   where $\mathbf{w}$ is a vector with all the elements of the weight set W as its components, k is the time index, at which a pair $(\mathbf{d}_k, \mathbf{x}_k)$ randomly comes from the training set D_{tr}. Moreover, ϵ_k is considered as a zero-mean stationary Gaussian noise sequence.

   Suppose $\mathbf{w}_k$ is an estimate of $\mathbf{w}$ at time k, we linearize $F(.)$ about $\mathbf{w}_k$ as follows:

$$F(\mathbf{x}_k, \mathbf{w}) \approx F(\mathbf{x}_k, \mathbf{w}_k) + H_k^t(\mathbf{w} - \mathbf{w}_k)$$
$$H_k = \frac{\partial F(\mathbf{x}_k, \mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}_k}. \tag{40}$$

   Thus, we have

$$\epsilon_k = \mathbf{d}'_k - H_k^t \mathbf{w}$$
$$\mathbf{d}'_k = \mathbf{d}_k - F(\mathbf{x}_k, \mathbf{w}_k) + H_k^t \mathbf{w}_k . \tag{41}$$

   We let the next $\mathbf{w}_{k+1}$ be estimated by the

minimization of the following weighted least square error

$$E_r^2 = \frac{1}{k}\sum_{l=1}^{k}\alpha_l||\epsilon_k||^2 = \frac{1}{k}\sum_{l=1}^{k}\alpha_l(\mathbf{d}_l' - H_l^t\mathbf{w}_{k+1})2\,, \tag{42}$$

which results in

$$\begin{aligned}\mathbf{w}_{k+1} &= R_k^{-1}\sum_{l=1}^{k}\alpha_l H_l\mathbf{d}_l' \\ R_k &= \sum_{l=1}^{k}\alpha_l H_l H_l^t\,,\end{aligned} \tag{43}$$

where $\{\alpha_l\}$ are appropriate weighting factors. Furthermore, by some simple manipulation, Eq. (43) can be rewritten into the following recursive version

$$\begin{aligned}R_k &= R_{k-1} + \alpha_k H_k H_k^t \\ \mathbf{w}_{k+1} &= \mathbf{w}_k + \alpha_k R_k^{-1} H_k[\mathbf{d}_k - F(\mathbf{x}_k, \mathbf{w}_k)]\,.\end{aligned} \tag{44}$$

The result obtained by Eq. (44) is usually called *recursive least square estimate* of the parameters of the nonlinear system function $F(\mathbf{x}, \mathbf{w})$.[53] Observing the definition of the matrix H_k in Eq. (40), we see that its elements can either be obtained directly by differentiation of each weight (if the weight belong to the output layer) or by using the chain rule in a way similar to Eqs. (9) and (10) (if the weights belong to hidden layers). Moreover, by using the matrix recursive inversion lemma to R_k^{-1} and by defining $P_k = R_k^{-1}$ and $\alpha_l = \lambda^{k-l}, 0 < \lambda \leq 1$, one can obtain the *Extended Kalman filter* algorithm as follows:

$$\begin{aligned}G_k &= P_{k-1}H_k \\ K_k &= G_k(\lambda I + H_k^t G_k)^{-1} \\ P_k &= \frac{1}{\lambda}(P_{k-1} - K_k G_k^{-1}) \\ \mathbf{w}_{k+1} &= \mathbf{w}_k + K_k[\mathbf{d}_k - F(\mathbf{x}_k, \mathbf{w}_k)]\,,\end{aligned} \tag{45}$$

which was first applied for training multilayer perceptrons in Ref. 205. Although at each step both Eqs. (44) and (45) will spend more computations than backpropagation takes, it has been reported that they can speed up training and improve convergence.[205,53] Some variants of Eq. (45) have also been proposed. In Ref. 172, weights are divided into groups (e.g. in Ref. 172 either the weights of each neuron was considered or simply just each weight as a group) and it is assumed that the elements of P_k corresponding to weights from different groups can be ignored. Thus, by re-ordering the components of $\mathbf{w}$, P_k can be arranged into a block diagonal matrix. As a result, one can use Eq. (45) separately on each group with reduced dimensions (i.e. the dimension of each block). This variant can reduce computation complexity while it does not seriously affect performance. In Refs. 196, 190 and 118, each neuron $o_j^{(q)} = \sigma(\mathbf{o}^{(q-1)}, \mathbf{w}_j^{(q)}) = \sigma(\sum_{k=1}^{n_{(q-1)}} w_{jk}^{(q)} o_k^{(q-1)})$ is treated as an independent system function and it is assumed the desired output of the neuron is given by $d_j^{(q)} = o_j^{(q)} + e_j^{(q)}, e_j^{(q)} = -\frac{\partial E_2}{\partial o_j^{(q)}}$. Thus from $e_j^{(q)} = d_j^{(q)} - \sigma(\mathbf{o}^{(q-1)}, \mathbf{w}_j^{(q)})$, in a way similar to that used for treating Eq. (39), one can obtain a recursive least square estimate formula for $\mathbf{w}_j^q$. The method, called independent extended Kalman filter, can further reduce computational complexity. However, it can lead to unstable behavior during training and will often result in solutions that are inferior to those found by either Eq. (45) or backpropagation.[172]

2. *Linear programming method.*
The idea is to find a weight change matrix δW such that for every pair $(\mathbf{d}_p, \mathbf{x}_p)$, the following condition is satisfied:

$$\begin{aligned}\delta(e_i^p)^2 = &[f_i(\mathbf{x}_p, W+\delta W) - \mathrm{d}_{pi}]^2 \\ &-[f_i(\mathbf{x}_p, W) - \mathrm{d}_{pi}]^2 \leq 0, i = 1, \ldots, m\end{aligned} \tag{46}$$

and such that at the same time the $\delta(e_i^p)^2, i = 1, \ldots, m$ are as small as possible. By taking a local first order approximation to this inequality, we have

$$\sum_{\text{all } w_{jk}^{(q)} \in W} \frac{\partial\delta(e_i^p)^2}{\partial w_{jk}^{(q)}}\delta w_{jk}^{(q)} \leq 0\,, \quad i = 1, \ldots, m\,, \tag{47}$$

where $e_i^p = f_i(\mathbf{x}_p, W) - \mathrm{d}_{pi}$ and $\delta w_{jk}^{(q)}$ is the change on the weight $w_{jk}^{(q)}$. Furthermore, the weight changes must be restricted to minimize the second-order effects $|\delta w_{jk}^{(q)}| \leq \varepsilon_{jk}^{(q)}$, i.e.

$$\begin{aligned}\delta w_{jk}^{(q)} &\leq -\varepsilon_{jk}^{(q)} \\ -\delta w_{jk}^{(q)} &\geq -\varepsilon_{jk}^{(q)}\,,\end{aligned} \tag{48}$$

where $\varepsilon_{jk}^{(q)}$ are prespecified non-negative small

quantities.

In summary, the problem of deciding appropriate values of $\delta w_{jk}^{(q)}$ for all j, k, q has become the following constrained optimization problem,

$$\text{Min}_{\delta w_{jk}^{(q)}} \sum_{p=1}^{P} \sum_{\text{all } w_{jk}^{(q)} \in W} \frac{\partial \delta (e_i^p)^2}{\partial w_{jk}^{(q)}} \delta w_{jk}^{(q)}, \quad (49)$$

subject to inequalities Eqs. (47) and (48) for all j, k, q, p where all the $\frac{\partial \delta (e_i^p)^2}{\partial w_{jk}^{(q)}}$'s can be calculated just from backpropagation formula Eqs. (9) and (10). Thus, Eqs. (49), (47) and (48) are all linear with respect to $w_{jk}^{(q)}$ and the above optimization problem is just a linear programming problem which can be solved by a number of algorithms available in the mathematical programming literature. According to Refs. 198, the method can avoid local minima, improve convergence and can as well give better degradation properties. The disadvantage is that each step will take a large amount of calculation. However, the total computation may be not so large since much less steps are needed in comparison with that spent by backpropagation.

3. *Fixing one layer while updating another.*
This is a common policy for simplifying the problem and speeding up calculations and it is usually used for training a net with only one hidden layer. There are a number of algorithms based on this policy. Here we give four major examples as follows;

   (1) In an algorithm called BRD given in Ref. 66, the weights of the hidden layer are randomly set and they are fixed when the weights of the output layer are trained by the simple perceptron algorithm. After being trained, the misclassification rate of this net is checked to see whether it is below the prerequired threshold. If not, another set of values are randomly generated to set the weights of the hidden layer and the above same schedule is repeated again until the misclassification rate fits our satisfaction.

   (2) In Ref. 1, the weights of the output layer are first fixed and the weights of the hidden layer are determined through the solution of the following equations by the Newton-Raphson method;

$$\frac{\partial E_2}{\partial w_{jk}^{(1)}} = 0, \quad \text{for all } i, j. \quad (50)$$

Then the weights of the hidden layer are fixed, the weights of the output layer are decided through the solution of the following equations again by the Newton-Raphson method

$$\frac{\partial E_2}{\partial w_{jk}^{(2)}} = 0, \quad \text{for all } i, j. \quad (51)$$

The same calculations are iterated for the weights $w_{jk}^{(1)}, w_{jk}^{(2)}$ respectively until a convergence is reached.

   (3) In Ref. 208, for each training pair $(\mathbf{d}_p, \mathbf{x}_p)$, first the weights of the output layer are modified according to the simple least square method for one layer case since the derivatives for this layer are directly obtainable. After that, a hidden target vector $\mathbf{o}_d^{(1)} = [o_1^d, o_2^d, \ldots, o_{n_1}^d]$ is estimated by

$$\text{Min}_{\mathbf{o}_d^{(1)}} \|\mathbf{d}_p - W^{(2)} \mathbf{o}_d^{(1)}\|^2, \quad (52)$$

where $W^{(2)}$ denotes the matrix consisting of all the current weights of the output layer. Next, the weights of the hidden layer can be modified in the same way as those of the output layer by using $\mathbf{o}_d^{(1)}$ as the desired output of the hidden layer.

   (4) Reference 72 suggested another one-pass training method for the problem with binary desired output. The basic ideas are as follows: assume that the nonlinear mapping between the input $\mathbf{x}$ and output $o_j^{(1)}$ given by a hidden neuron can be approximated by

$$X \mathbf{w}_j^{(1)} = K_j^{(1)} \mathbf{d}_j^{(1)}, \quad j = 1, \ldots, n_1, \quad (53)$$

where

$$X = [\mathbf{x}_1, \ldots, \mathbf{x}_p]^t, \quad \mathbf{d}_j^{(1)} = [d_{1j}^{(1)}, \ldots, d_{pj}^{(1)}]^t$$

is a unknown vector but with the constraint $d_{pj} \in \{-1, 1\}$ and $K_j^{(1)}$ is a $p \times p$ unknown diagonal matrix. Singular value decomposition analysis is used on X to decide, for each hidden neuron, one solution of $\mathbf{w}_j^{(1)}$ which satisfies Eq. (53). After that, X can be propagated forward into its

corresponding output matrix D_h of hidden neurons. Similarly, assume that the nonlinear mapping of an output neuron can be approximated by

$$D_h \mathbf{w}_j^{(2)} = K_j^{(2)} \mathbf{d}_j^{(2)}, \quad j = 1, \ldots, n_1, \tag{54}$$

with $\mathbf{d}_j^{(2)} = [d_{1j}^{(2)}, \ldots, d_{pj}^{(2)}]^t$ being a binary vector which is determined from the given desired output vector $\mathbf{d}_p, p = 1, \ldots, P$. Then SVD is again used on D_h to determine, for each output neuron, one solution of $\mathbf{w}_j^{(2)}$ which satisfies Eq. (54).

4. *Networks converted from decision tree classifiers.*

Recently several authors have studied the connections between multilayer perceptrons and the conventional decision tree classifiers (Refs. 194, 195, 27 and 218). It has been shown that a net can be built by converting a special kind of binary decision tree into a multilayer architecture.

In this kind of binary decision trees, each of all the nonterminal nodes is a simple Perceptron or Adline with the whole input vector $\mathbf{x}$ as its input (i.e. each node defines a hyperplane $w_0 + \mathbf{w}^t\mathbf{x} = 0$ which divides the input pattern space into two halves). Each leaf node corresponds to the class that $\mathbf{x}$ is finally classified into. Such a binary decision tree can be trained by a number of classical techniques in the literature of statistical pattern recognition and statistical decision theory (Refs. 174, 104 and 26).

After being trained, it can be easily converted into a two-hidden-layer feedforward network. All the nonterminal nodes of a binary tree are converted into the neurons of the first hidden layer. As a result, each neuron in the first layer is fully connected to the input $\mathbf{x}$ and its weight vector is just the parameter vector of the hyperplane defined by the corresponding nonterminal node in the binary tree. Furthermore, each path in the decision tree from the root node to a leaf node is converted into a neuron of the second hidden layer. Each of these neurons is only connected to those neurons of the first hidden layer which are corresponding to the nodes located on this path of the binary tree and the weight of each connection is either $+1$ or -1 depending on whether the path passes the left or right branch from the node corresponding to the connected neuron in the first hidden layer to this node's two children. Finally, each neuron in the output layer corresponds to a single output class and it connects all the neurons of the second hidden layer with weight $+1$ if each of these neurons corresponds to a node on the path with its leaf node representing this pattern class. In Ref. 194, the second hidden layer is called the *ANDing* layer since each of its neurons computes a conjunction of its inputs or their negatives; the output layer is called the *ORing* layer since each of its neurons computes a disjunction of its inputs.

After the above conversion, the resulted net can either be directly used as classifier or as a good starting approximation for other training techniques such as backpropagation. One advantage of this way of constructing a net is that, in comparison with backpropagation, the training is much faster and the performance can be improved substantially[195,27]. In addition, through this channel various classical techniques for designing tree classifiers can be introduced into the literature of neuron networks.

5. *Construction of networks based on perceptron learning.*

There is recently a seemingly renewed interest in using perceptron learning rules. The decision tree related method, described above, is one such example. There are also a number of attempts of using perceptron learning to construct a net forward layer-by-layer (Refs. 114, 63 and 145). These attempts share a common point: to gradually add neurons (or equivalently hyperplanes) into a net such that parts of training patterns can be correctly classified by some single hyperplanes and others can be correctly classified by combinations of a number of hyperplanes until all the training patterns are correctly classified. One example of such attempts is given by Ref. 114 in which each neuron is gradually added into the first hidden layer in such an order that linear separation of each class is tried first and the linear separation of pairs of classes is tried after, then a binary decision tree is tried to introduce successively hyperplanes (i.e. neurons) until all the remaining training patterns are rightly classified. Hereafter, one or more

additional layers are externally designed to perform explicit boolean functions that will combine appropriately all the hyperplanes to form the final output layer. Other two examples are the *tiling algorithm*[145] and the *upstart algorithm*.[63]

Generally speaking, there are usually a great number of ways to split an input pattern space by gradually introducing hyperplanes, the optimal design of hyperplanes is identical to the optimal design of a tree decision classifier with each node being a hyperplane. Thus, the heuristic forward constructed algorithms given by Refs. 114, 63 and 145 are closely related to the design of tree classifiers. From this aspect, the way of constructing a net to convert a binary decision tree classifier might be a better choice for networks construction than these heuristic algorithms, since there are already a number of sophisticated methods for suitably designing tree classifiers and the process of converting a tree classifier into a net is pretty simple.

4. Other Models for Feedforward Networks

After several years of extensive studies on multilayer perceptron, researchers' attentions have been turned to a number of other models for feedforward networks, especially in the recent two years. For convenience, here we group these models into three big categories. The first and the biggest one consists of various models which implement function approximation or probability density estimation by *basis function expansion* (Refs. 64, 95, 12, 65, 161, 10, 61, 173, 164, 175, 176, 192, 133, 98, 193, 209, 210, 37, 62, 171, 28, 169, 39, 68, 150, 177, 121, 158, 213, 186, 247, 166, 233, 107, 105 and 24), i.e. the networks function given by Eq. (3) is replaced by

$$f_i(\mathbf{x}) = \sum_{j=1}^{N} w_{ij}\phi_j(\mathbf{x}), \quad i = 1, \ldots, m. \tag{55}$$

Represented in the network formalism, the functions correspond to a one-hidden-layer architecture with N hidden neurons. Each hidden neuron is fully connected to every components of input $\mathbf{x}$ and represents a basis function $y_i = \phi_j(\mathbf{x})$. The output layer has m neurons with each being a linear summation neuron $\phi_i = \sum_{j=1}^{N} w_{ij} y_i$ with weight vector $\mathbf{w}_i = [w_{i1}, \ldots, w_{im}]^t$. Apparently, this architecture can be regarded as a variant of one-hidden layer perceptron by changing the hidden neuron's functions, as we previously discussed in Sec. 2.5. However, there is a key difference that the updating of function $\phi_j(\mathbf{x})$ is no longer decided by the chain rule based supervised learning but predetermined externally or specified directly by training samples or developed by some self-organizing techniques. Thus we would like to consider them as a group of new models.

The second category collects several other supervised learning models of feedforward networks, such as Kohonens learning vector quantization algorithm,[117,116] feedforward Boltzman machine,[155] query learning (Refs. 19, 99 and 249) etc.

The third category consists of models which have various complex structures: hierarchies, modules and others which contain some of the previously reviewed nets (e.g. multilayer perceptrons) as building blocks in order to deal with more complicated problems or to gain some specific advantages (Refs. 103, 159, 244, 214, 187, 160, 23, 40, 16 and 25).

In the sequel, we will survey the models of the first category in Sec. 4.1 and the last two categories in Sec. 4.2.

4.1. *Models related to various basis functions*

Given a training set $D_{tr} = \{\mathbf{d}_p, \mathbf{x}_p, p = 1, \ldots, P\}$, the weights $w_{ij}, i = 1, \ldots, m, j = 1, \ldots, N$ of the output layer can be easily determined if the basis functions $y_j = \phi_j(\mathbf{x}), j = 1, \ldots, N$ are known *a priori*, i.e. $W = [w_{ij}]_{m \times N}$ can be obtained by minimizing the following least square error E_2

$$\begin{aligned} E_2 &= \sum_{p=1}^{P} \|\mathbf{d}_p - W\mathbf{y}_p\|^2 \\ \mathbf{y}_p &= [y_1^{(p)}, \ldots, y_N^{(p)}]^t \\ y_j^{(p)} &= \phi_j(\mathbf{x}). \end{aligned} \tag{56}$$

We can tackle the minimization problem either in *block way* by directly solving a linear equation derived from Eq. (56), or in *on line way* by the following simple gradient updating rule

$$W(t+1) = W(t) + \alpha(\mathbf{d}_p - W\mathbf{y}_p)\mathbf{y}_p^t. \tag{57}$$

Thus, the key point of building a basis function network is to appropriately select basis functions $\phi_j(\mathbf{x}), j = 1, \ldots, N$. There are various ways for such kind of selection which result in different models. Here we roughly divide these models into two groups

according to the types of basis functions presently used in the neural network literature. The first group is called localized basis functions. That is, the basis functions of the models in this group can be expressed in the following general form

$$\phi_j(\mathbf{x}) = \phi(\mathbf{x} - \mathbf{c}_j, \Sigma_j), \quad j = 1, \ldots, N, \qquad (58)$$

where $\phi(.)$ is called a *mother function*. Each basis function is obtained by locating the mother function at a point of pattern space given by the locating parameter vector $\mathbf{c}_j$ and may or may not be subject to some deformation caused by an $n \times n$ parametric matrix Σ_j. The second group is called nonlocalized basis functions which are not expressible by Eq (58).

For convenience, we start at the review of the models in the second group since they are relatively easier to describe in a smaller space. The models given in the followings are several typical examples of this group:

- *Polynomial related basis functions.*
  In the simplest case, $\phi_j(\mathbf{x})$ is directly chosen as canonical polynomial terms $x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n}, r_i \geq 0$; i.e., $\phi_0 = \text{const.}\ \phi_j = x_j, j = 1, \ldots, n, \phi_j = x_i x_k, i, k = 1, \ldots, j = n+1, \ldots, n^2 + n + 1, \ldots$, etc. suggested by Refs. 161, 210 and 37. Furthermore, $\phi_j(x)$ can be some sophisticated polynomial, e.g. in Ref. 10, Bernstein polynomials are used as basis functions for the special problems of approximating a 1-dimensional function; and a more general example is given in Ref. 61 where an infinite sum of polynomials called reproducing kernels are used as $\phi_j(\mathbf{x})$ i.e.

$$\phi_j(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}_j) = \sum_{r=0}^{\infty} \frac{1}{r! \rho_r} (\mathbf{x}^t \mathbf{x}_j)^r, \qquad (59)$$

  with $\mathbf{x}_j$ being a training sample and ρ_r being a predefined sequence of positive number satisfying a specified condition.
- *Trigeometry basis net.*
  When $\mathbf{x} = x$ is one dimensional, the simple functions $\sin(j\pi x), \cos(j\pi x), j = 0, \ldots, N$ can be directly used as $\phi_j(x)$[161] while when $\mathbf{x} \in R^2$, cosine bases $\cos(i\pi x_1)\cos(j\pi x_2), i = 1, \ldots, N$, $j = 1, \ldots, N$ are suggested in Ref. 173. Moreover, they also take the probability density $p(\mathbf{x})$ in consideration and suggest to first transform $\mathbf{x}$ into $\mathbf{u} = [\mu_1, \mu_2]^t, \mu_1 \mu_2 = p(\mathbf{x}) dx_1 dx_2$ and then the above cosine basis are used in the space (μ_1, μ_2). However, it seems that this transform is not practically useful since it is usually quite hard to know $p(\mathbf{x})$ priori.
- *Projection pursuit.*
  This is a nonparametric statistical regression model (Refs. 64, 95 and 12). In this model, all the weights w_{ij}'s in the output layer are fixed at constant 1 while basis functions are given by $\phi_j(\mathbf{x}) = g_j(\mathbf{x}^t \theta_j)$ where $\theta_j, j = 1, \ldots N$ are normalized unit length vectors for representing N projection directions and $g_j, j = 1, \ldots, N$ are one-dimensional nonparametric functions. Both the directions θ_j's and functions g_j's should be determined empirically from training data set, obtained by using an algorithm called backfitting. Using this algorithm, we initialize $g_j = 0, \theta_j = \theta_j^0, j = 1, \ldots, N$, then for each j, find first a one-dimensional smooth function $s(\mathbf{x}^t \theta_j)$ as the new g_j which "best" fits $r_j(\mathbf{x}) = f(\mathbf{x}) - \sum_{i=1, i \neq j}^{N} g_i(\mathbf{x}^t \theta_i)$ and then solve for a new direction θ_j by minimizing $\sum_{p=1}^{P} [r_j(\mathbf{x}_p - g_j(\mathbf{x}_p^t \theta_j)]^2 / \sum_{p=1}^{P} r_j^2(\mathbf{x})$. This process is iterated until convergence. In order to estimate g_j, techniques such as cubic splines can be used. Recently, Ref. 254 used this kind of learning technique to learn 2-Joint robot arm dynamics and reported reasonable results. This seems to be an interesting model to be further investigated.
- *Basis function tree.*
  Polynomials have a natural representation as a tree structure. In this representation, the output of a subtree determines the weight from that node to its parent. For example, $f(x, y) = ax^2 + bxy + cy$ can be expressed into $f(x, y) = A(x, y)x + cy = (ax + by)x + cy$. In this tree representation, we have a weight $A(x, y)$ on the link from node x to the root $f(x, y)$ and a weight c from node y to the root $f(x, y)$. Furthermore the weight $A(x, y)$ is the output of a subtree of two nodes x and y in which there are a weight a from node x to $A(x, y)$ and a weight b from node y to $A(x, y)$. Based on this representation, Ref. 188 proposed an algorithm called the "LMS tree" to approximate functions by separable basis functions. That is, assuming that there are a finite set of one variable function $\{s_r\}_{r=1}^{n_s}$ and each basis function $\phi_j(\mathbf{x})$ given in Eq. (55) can be written into

$$\phi_j(\mathbf{x}) = s_{r_1^j} \cdots s_{r_n^j}(x_n), \qquad (60)$$

  where x_p is the pth component of $\mathbf{x}$, $s_{r_p^j}(x_p)$ is a scalar function of scalar variable x_p chosen from the set $\{s_i\}_{i=1}^{n_s}$ (i.e., r_p^j is an integer between [1, n_s]). In this case, one can regard $f_i(\mathbf{x})$ given in Eq. (55) as a n-order polynomial of $n_s n$ variables

in terms of $\{s_r(x_k)\}_{r=1}^{n_s}, k = 1,\ldots,n$. Thus, the function can be expressed in the form of a tree of depth n. The tree can have at most n_s^n number of nodes. The building of such a tree is very hard for large n_s and n. Reference 188 proposed a solution. It first arbitrarily select one variable (say x_1) for building the first layer

$$f_i(\mathbf{x}) \approx f_i(x_1) = \sum_{r=1}^{n_s} w_r^{(1)} s_r(x_1). \qquad (61)$$

The parameters $w_r^{(1)}$'s are decided through the Widrow-Hoff LMS learning rule[240] with a constant learning rate. The learning will reach its balance with $E(\Delta w_r^{(1)}) = 0$ but the variance $\mathrm{Var}(w_r^{(1)}) \neq 0$ due to the constant learning rate. Thus, in the next step, the largest of these variance is selected and the corresponding parameter (say $w_{r*}^{(1)}$) is further approximated by one subtree of the second layer

$$w_r^{(1)} = E(w_{r*}^{(1)}) + \sum_{r=1}^{n_s} w_r^{(2)} s_r(x_2), \qquad (62)$$

which use a new variable x_2. Here, the error $E(\Delta w_{r*}^{(1)})$ can be used to determine the parameters $w_r^{(1)}$'s by LMS rule again. In Ref. 188, new subtrees would include all dimensions and could be grown below any $s_r(x_p)$. Thus, a very large tree will usually be obtained. It is suggested that weights below a threshold level are set to zero and any leaf with zero weight can be removed.[188] It is reported that the algorithm has a very high learning speed and a very low storage requirement, and it can also adapt to slowly-varying nonstationary environments.

Now, let us proceed to the first group of models — localized basis functions. Studies of these models have been quite vigorous in the past two years and probably it forms the second focus point of supervised learning nets after backpropagation. The typical models includes Multivariate Adaptive Regression Splines (MARS) nets,[65] Wavelet function nets,[164] Restricted Coulomb Energy (RCE) nets (Refs. 175, 176, 133, 98, 192 and 193), Radial Basis Function (RBF) nets and variants (Refs. 62, 171, 169, 150, 28, 39, 68, 177, 121, 158, 105, 213, 186, 247, 166, 233, 107 and 24). These models are generating an ever-increasing interest in the literature.

In the sequel, these models are further reviewed in detail.

- *Multivariate Adaptive Regression Splines (MARS) nets.*

The basic idea is to modify the tensor-product spline method for regressing or approximating multivariate functions into some adaptive version such that the high computational costs can be reduced considerably.[65] Similar to Eq. (55), a function can be approximated by the combination of spline basis as

$$f_i(\mathbf{x}) = \sum_l w_{li} B_l(\mathbf{x}), \quad i = 1,\ldots,m, \qquad (63)$$

where the basis function set $\{B_l(\mathbf{x})\}$ is obtained by taking the tensor product of the set of q order spline functions $\{x_j^i\}_{i=0}^q, \{(x_j - t_{kj})_+^q\}_{k=1}^K$[f] over all the axes $j = 1,\ldots,n$ of input vector $\mathbf{x} \in R^n$. That is, each of the $K+q+1$ functions on each axis j is multiplied by all of the functions corresponding to all of the other axes $k, k = 1,\ldots,n, k \neq j$. As a result, the total number of basis functions are $(K+q+1)^n$ which is usually a huge number. Each basis function is located on a specific one of the regions partitioned by knots $t_{kj} (k = 1,\ldots K, j = 1,\ldots,n)$.

This simple tensor-product spline basis function have severe limitations due to computational costs that preclude their use for high dimensional data ($n \gg 2$). To overcome the limitations, the key idea of MARS is to select a relatively small subset among the huge set of $(K+q+1)^n$ basis functions. The particular subset for a problem at hand is obtained through standard statistical variable subset selection, treating the basis functions as the "variables". At the first step the best single basis function is chosen. The second step chooses the basis function that works best in conjunction with the first. At the mth step, the one that works best with the $m-1$ already selected, is chosen and so on. The process stops when including additional basis function fails to improve the approximation.

In Ref. 65, a network implementation that approximates the above adaptive spline strategy is described. It is equivalent to selecting a subset of basis functions in such a way that at the beginning, $k = 0, B_0(\mathbf{x}) = 1$ is chosen as one basis function, then at the $k = 1$ step, choose

[f] $(x_j - t_{kj})^q$ is known as the truncated power function, i.e. $(x_j - t_{kj})_+^q$ is 0 when $x_j \leq t_{kj}$ and $(x_j - t_{kj})^q$ when $x_j > t_{kj}$ and there is one for each knot location $t_{kj} (k = 1,\ldots K)$ on each input axis $j (j = 1,\ldots,n)$.

one x_j from the set $x_j, j = 1, \ldots, n$ to produce the second and the third basis functions $B_1(\mathbf{x}) = B_0(\mathbf{x})(x_j - t_k)_+^q, B_2(\mathbf{x}) = B_0(\mathbf{x})(t_k - x_j)_+^q$. Furthermore at the $k = K$ step, choose one x_j from the $x_j, j = 1, \ldots, n$ and choose B_{r_k} from the $B_k, k = 1, \ldots, 2K + 1$ to form another pair of basis functions $B_{2K+2}(\mathbf{x}) = B_r(\mathbf{x})(x_j - t_K)_+^q$, $B_{2K+3}(\mathbf{x}) = B_r(\mathbf{x})(t_K - x_j)_+^q$. The process is repeated until k reaches its maximum $M_{\max}$. During the process, at each k, there are three parameters r_k, j_k, t_k that need to be determined. The goal of training the network is to choose values for these parameters by the following generalized cross-validation criterion

$$\mathrm{GCV} = \frac{1}{mN}\sum_{i=1}^{m}\sum_{p=1}^{N}(d_i(\mathbf{x}_p) - f_i(\mathbf{x}_p)^2)\Big/\left(1 - \frac{5k+1}{N}\right)^2, \tag{64}$$

with $(\mathbf{d}_i(\mathbf{x}_p), \mathbf{x}_p), p = 1, \ldots, N$ being the training set. The training strategy used is a semi-greedy one. At step K, the GCV criterion is minimized only with respect to the present r_k, j_k, t_k and the new weighting parameters in Eq. (63) correspond to the two new basis functions B_{2K+2}, B_{2K+3} with all the pervious parameters fixed. This optimization can be done very quickly. The total computation is $O(nNM_{\max}^3)$. Although this method is obviously suboptimal, experiments conducted by Ref. 65 show that the better optimization seldom resulted in even a moderate improvement. The reason is that the basis functions added later can compensate for the suboptimal settings of parameters introduced earlier.

One interesting property of MARS is that it unifies *additive function and CART* models into a single framework. Both additive and CART approximations have been highly successful in largely complementary situations: additive modeling when the true underlying function is close to additive and CART when it dominantly involves high order interactions between the input variables. It is hoped that MARS will be successful at both these extremes as well as the broad spectrum of situations between where neither works well. It seems that further experimental works and applications are required to be made to test the new model.

- *Wavelet function net.*

Given a function $y = f(x)$ of one real variable and assuming $f \in L^2(R)$ the space of square integrable functions on the real line, then there exists a convergent series representation for $f(x)$

$$f(x) = \sum_m \sum_n c_{mn}\psi_{mn}(x), \tag{65}$$

where basis functions are of form

$$\psi_{mn}(x) = a^{-m/2}\psi(a^{-m}x - nb), \tag{66}$$

with the function ψ satisfying appropriate admissibility conditions (e.g. $\int \psi = 0$) and suitable choices of $a > 1, b > 0$. Here, the representation of f by Eq. (65) as a series in dilatations and translations of a single function ψ is called a wavelet expansion and the function ψ is known as the analyzing or mother wavelet for the expansion. Given appropriate ψ and a, b, the expansion parameters c_{mn} can be computed in a way similar to that of solving w_{ij} in Eq. (55) discussed in Sec. 4.1. The key idea of a wavelet net is, based on a training set, to appropriately select a mother wavelet ψ and a, b as well as a finite subset $\Psi_f = \{\psi_{mn}\}$ such that f is approximated by the truncated expansion $\sum\sum_{\psi_{mn} \in \Psi_f} c_{mn}\psi_{mn}(x)$. In Ref. 164, ψ is defined as $\psi(x) = \sigma(x + 2) + \sigma(x - 2) - 2\sigma(x), \sigma(x) = 1/(1 + e^{-x})$ and then a, b are chosen appropriately. Furthermore, Ψ_f is decided according to the so-called spatio-spectral concentrations of wavelet Ψ_{mn} and of f. Wavelet expansions for functions in $L^2(R^n)$ are also possible,[142] which also makes it possible to build wavelet nets for approximating multivariate vector-valued functions.[164] Presently, the direction is rather new and more studies are needed.

- *Restricted Coulomb Energy (RCE) nets.*

The nets are used specifically for classification purpose, i.e. in the case that given a training pair $\{\mathbf{d}_p, \mathbf{x}_p\}$ and $\mathbf{d}_p = [d_{p1}, \ldots, d_{pm}]^t$, we have $d_{pi} = 1$ if $\mathbf{x}$ belongs to class i, otherwise, $d_{pi} = 0$. In a RCE net, the localized basis function used here is very simple, given by $\phi_j(\mathbf{x}) = u(R_j^2 - \|\mathbf{x} - \mathbf{c}_j\|^2)$ where $u(r)$ is a step function i.e. $u(r) = 1, r \geq 0, u(r) = 0, r < 0$. In other words, each basis function specifies a hypersphere in the pattern space which is centered at $\mathbf{c}_j$ and has a parameter of radius R_j. If an input pattern $\mathbf{x}$ falls into the sphere, the neuron corresponds to the basis function outputs 1, otherwise, it outputs 0. In the output layer, there are m units each of which corresponds to a class label, the ith unit is activated if $f_i(\mathbf{x}) > 0$ where $f_i(\mathbf{x})$ is again given by Eq. (55). The training strategy is rather simple. In the beginning, there is no unit in the net. For the

first training pattern, a unit is added with $\mathbf{c}_1$ being equal to the pattern and R_1 with an arbitrary value. In the same time, a unit weight is given to connect the unit to the output-layer unit which corresponds to the class label of the pattern, i.e. suppose that the class label of the pattern is r, we have $w_{r1} = 1$ for the weights given in Eq. (55). After a number of training patterns, assume that there are k units in the net, then when a new training pattern with label r comes, there are two possible treatments:

(1) The pattern does not fall in any of the hyperspheres defined by these units, the $k+1$th unit is added with $\mathbf{c}_{k+1}$ being equal to the pattern and R_{k+1} being the minimum distance between the pattern to one existing unit which has unit weight connected to an output-layer-unit not corresponding to the class label r. In the same time, we set $w_{r(k+1)} = 1$ (i.e. connecting the new unit with unit weight to the rth unit of the output layer.)

(2) When the pattern falls in the hyperspheres defined by some units, for each of these units we check whether it has an unit weight connected to an output-layer unit corresponding to class label different from r. If yes, the radius of the unit is reduced such that the hypersphere is shrunken to not include the pattern; otherwise, nothing is done. This simple learning algorithm has been described in details by Refs. 175 and 176.

The RCE net has been studied by a number of authors (Refs. 175, 176, 133 and 98). It can be regarded as the modification of the hyperspherical classifiers developed in the 1960's by Refs. 42, 43, 14 and 15 where the center vectors $\mathbf{c}_j$'s can be moved and R_j's can be either reduced or increased. One main limitation of RCE net is that it works well only when the class regions are separable. However, when the class regions have some overlaps and a pattern comes from such overlapped regions. There will more than one unit in the output layer will be activated and the result conflicts. A remedy is given in Ref. 192, the idea is to let the weights w_{rj} being not simply 1 but the count of the patterns in the whole training set which fall in the hypersphere defined by the unit j. Recently, in Ref. 193, the spherical function $u(R_j^2 - ||\mathbf{x} - \mathbf{c}_j||^2)$ is further replaced by $\exp(-||\mathbf{x} - \mathbf{c}_j||^2)$ and the output-layer-units' output is directly given by Eq. (55). This new modification of RCE makes it is quite similar to the RBF nets which will be surveyed in the following. The only difference is that in Ref. 193, the training still uses the simple learning algorithm given above while in RBF nets other learning methods are proposed. The recent experimental investigations on the RCE net by Refs. 133 and 98 also deserve mention. They have compared the performances of the RCE net, various back-propagation nets and some conventional clasifiers like nearest neighbour classifier.

- *Radial Basis Function (RBF) nets and variants.* The original model of RBF net is obtained simply by letting $\phi_j(\mathbf{x}) = \phi(||\mathbf{x} - \mathbf{x}_j||)$, i.e. in Eq. (58), let $\Sigma_j = I$ and $\phi(.)$ be radial symmetric around the locating center $\mathbf{c}_j = \mathbf{x}_j$ where $\mathbf{x}_j$ is a training sample. Thus, Eq. (55) can be rewritten into

$$f_i(\mathbf{x}) = \sum_{j=1}^{P} w_{ij}\phi(||\mathbf{x} - \mathbf{x}_j||), i = 1, \ldots, m. \quad (67)$$

(Note: Here all the samples in the training set have been used.)

There are many possibilities for selecting $\phi(.)$. The most common one is Gaussian function $G(r) = \exp(-r^2)$ but a number of alternatives can also be used (Refs. 171, 28, 169 and 39). All these basis functions may give good results on a training set but may be different obviously on a testing set. In other words, we need to select suitable basis function according to problems in order to get good generalization ability.[24] It has been shown that the RBF expansion given in Eq. (67) is equivalent to the solutions derived from *regularization theory*, i.e. least square fitting subjected to a rotational and translational constraint term by a different operator.[169,253] It has also been shown[68] that this kind of RBF net has not only universal approximation ability but also best approximation ability.

However, the simple RBF net given by Eq. (67) has the serious disadvantage that all the training samples, which could be a great number, are used. This will not only cause considerable costs both for computing and storage but also makes the solution W of the output layer weights obtained by Eq. (56) ill-posed, unrobust and of poor generalization ability. The modification for remedying the problem have been made along two roads. The first road

is to try to select a subset of the whole training set D_{tr} by discarding those not so important samples so that the number P can be reduced to a much smaller number $K \ll P$. One solution for this task is the multiedit algorithm,[121] a method earlier developed for *nearest neighbor classifier*.[51] The other method along the first road is to let the space spanned by $\Phi = [\phi_{ij}], \phi_{ij} = \phi(||\mathbf{x}_i - \mathbf{x}_j||^2)$ be expressed by a set of orthonormal basis vectors and then to select a subset of these orthonormal basis vectors incrementally such that the total residuals, which result from the projections of the desired outputs $\mathbf{d}_p, p = 1, \ldots, P$ onto the subspace spanned by the subset of the orthonormal basis vectors are as small as possible.[39]

The second road is to modify Eq. (67) into

$$f_i(\mathbf{x}) = \sum_{j=1}^{n_h} w_{ij}\phi_j(||\mathbf{x} - \mathbf{c}_j||^2), \quad i = 1, \ldots, m, \tag{68}$$

i.e. to use a few movable location vectors to replace the direct use of training samples as the locations of each basis functions. In this case, the key point is how to determine those $\mathbf{c}_j, j = 1, \ldots, n_h$ which can be regarded as the weight vector of n_h hidden neurons. It is possible to use gradient descent updating $\delta\mathbf{c}_j = -\alpha\frac{\partial E_2}{\partial \mathbf{c}_j}$ through a backpropagation-like chain rule.[169] But, Ref. 150 found that such kind of learning is very slow. Instead, they use a modified *K-mean clustering* algorithm to find n_h cluster centers as $\mathbf{c}_j, j = 1, \ldots, n_h$ to speed up the learning. In Ref. 158, another version of the K-mean clustering algorithm is also used for training these hidden weight vectors. However, one problem of the K-mean clustering algorithm is that the number of clusters should be predefined externally. If this number is not appropriately chosen, the clustering results may be very poor and thus result in the poor performance of RBF net. Recently, a method called *Rival Penalized competitive learning* is proposed,[247] which can automatically decide an appropriate number of clusters for training data and the experiments have shown that it improves the performance of RBF considerably.[247] Another alternative which can locate these movable vectors $\mathbf{c}_j$ in an incremental way is proposed by Ref. 166. In the beginning, $\mathbf{c}_1$ is just the first sample, hereafter as a new sample comes, its distance to the nearest existing movable vector is calculated and the estimation error of the present RBF net is computed, if both the distance and the error are larger than the prespecified thresholds, a new movable vector is added in the net by simply letting it being the same as the sample; otherwise, all the existing movable vectors as well as weight w_ij's are modified to reduce the error in the gradient descent way through backpropagation by the chain rule.[169] This method will result in more movable vectors being used than the above K-mean type algorithm but the learning may become considerably faster.

The RBF expansion given by Eq. (68) can be further generalized into

$$f_i(\mathbf{x}) = \sum_{j=1}^{n_h} w_{ij}\phi([\mathbf{x} - \mathbf{c}_j]^t\Sigma_j^{-1}[\mathbf{x} - \mathbf{c}_j]) \quad i = 1, \ldots, m, \tag{69}$$

where Σ_j is a $n \times n$ semi-positive matrix. The simplest case is $\Sigma_j = \sigma_j^2 I_{n\times n}$ (Eq. (69) will reduce to Eq. (68) when $\sigma_j^2 = 1$). The parameters σ_j^2 affect the influential radius $||\mathbf{x} - \mathbf{c}_j||^2$ of every basis function located at $\mathbf{c}_j$ and sometime are called the width of the *receptive field* of the basis function. The value of σ_j^2 can be either prespecified which is actually equivalent to the case of Eq. (68) or determined by gradient descent updating $\Delta\sigma_j^2 = -\alpha\frac{\partial E_2}{\partial \sigma_j^2}$ through backpropagation-like chain rule[169] or some heuristic way jointly used with K-mean clustering algorithms.[158] Recently, Ref. 144 has also studied how σ_j^2's affects RBF net's learning and proposed some heuristic design strategies for adjusting these parameters as well as the number of hidden neurons.

One more general case for matrix Σ_j is that it is a diagonal matrix $\Sigma_j = \text{diag}[\sigma_{j1}^2, \ldots, \sigma_{jn}^2]$ i.e. the width of each basis function is scaled differently in every dimension. Similarly, Σ_j can be also determined by gradient descent updating $\Delta\sigma_{ji}^2 = -\alpha\frac{\partial E_2}{\partial \sigma_{ji}^2}, i = 1, \ldots, n$ through backpropagation-like chain rule.[169] But as noted in Ref. 233, the gradient descent way is easy to get trapped into local minima, thus they suggested to combine the estimation of their parameters with the procedure of using K-mean clustering algorithms for finding location centers $\mathbf{c}_j, j = 1, \ldots, n_h$. A heuristic formula for estimating these parameters was recently given in Ref. 24. It has also experimentally revealed that the appropriate selection of these scaling parameters do improve the generalization performance of RBF net considerably. The most general case for matrix Σ_j is that it is a nondiago-

nal semipositive matrix. This case is equivalent to the weighted norm case suggested in Ref. 169 and includes that studied in Ref. 186 as a special case when $\mathbf{c}_j = 0, j = 1, \ldots, n_h$.

There are also a number of other variants for RBF nets. In Ref. 69, the RBF net given by Eq. (68) has been extended to the cases that outlier and negative samples are also included in the given training set, by modifying the total error E_2 to consider the influences of outlier noises and negative samples based on the formalism of maximum posterior estimation. One other example is to replace Eq. (69) by

$$f_i(\mathbf{x}) = \sum_{j=1}^{n_h} w_{ij}\phi([\mathbf{x}-\mathbf{c}_j]^t\Sigma_j^{-1}[\mathbf{x}-\mathbf{c}_j])/ \times \sum_{j=1}^{n_h}\phi([\mathbf{x}-\mathbf{c}_j]^t\Sigma_j^{-1}[\mathbf{x}-\mathbf{c}_j]) \quad i=1,\ldots,m, \tag{70}$$

i.e. by normalizing the output of hidden neurons. This normalization is usually made when $f_i(\mathbf{x})$ is used for estimating probabilistic density function.[150,158] Moreover, for the special basis function $\phi(||\mathbf{x}-\mathbf{c}_j||^2) = \frac{1}{\sqrt{2\pi}\sigma_j}\exp(-||\mathbf{x}-\mathbf{c}_j||^2/2\sigma_j^2)$ it is shown[105] that Eq. (70) can be derived from a criterion which minimizes a least square cost function with the soft constraint that the mutual information between input and output is maximized. In Ref. 105, Eq. (70) is further replaced by

$$f_i(\mathbf{x}) = \sum_{j=1}^{n_h}[w_{ij} + (\mathbf{x}-\mathbf{c}_j)^t\mathbf{d}_j]\phi([\mathbf{x}-\mathbf{c}_j]^t \times \Sigma_j^{-1}[\mathbf{x}-\mathbf{c}_j])\Big/\sum_{j=1}^{n_h}\phi([\mathbf{x}-\mathbf{c}_j]^t \times \Sigma_j^{-1}[\mathbf{x}-\mathbf{c}_j]), \quad i=1,\ldots,m, \tag{71}$$

where Σ_j is a diagonal matrix

$$\Sigma_j = \mathrm{diag}[\sigma_{j1}^2, \ldots, \sigma_{jn}^2]$$

and $\mathbf{d}_j$ is an additional parameter vector. This variant is motivated by imposing the following constraint on $f_j(\mathbf{x})$

$$f_i(\mathbf{x}) = \sum_{j=1}^{n_h} f_j(\mathbf{x})\phi([\mathbf{x}-\mathbf{c}_j]^t \times \Sigma_j^{-1}[\mathbf{x}-\mathbf{c}_j])\Big/\sum_{j=1}^{n_h}\phi([\mathbf{x}-\mathbf{c}_j]^t \times \Sigma_j^{-1}[\mathbf{x}-\mathbf{c}_j]), \quad i=1,\ldots,m \tag{72}$$

and then expanding $f_i(\mathbf{x})$ by the first order of Taylor series about $\mathbf{c}_j$. Reference 213 also proposed a variant which is somewhat similar to Eq. (71), given as

$$f_i(\mathbf{x}) = \sum_{j=1}^{n_h}[w_{ij} + \mathbf{v}_{ij}^t\mathbf{x}_j]\phi([\mathbf{x}-\mathbf{c}_j]^t \times \Sigma_j^{-1}[\mathbf{x}-\mathbf{c}_j])\Big/\sum_{j=1}^{n_h}\phi([\mathbf{x}-\mathbf{c}_j]^t \times \Sigma_j^{-1}[\mathbf{x}-\mathbf{c}_j]), \quad i=1,\ldots,m. \tag{73}$$

One difference here is that $\mathbf{d}_j$ is replaced by $\mathbf{v}_{ij}$ which may be different from the connections between a hidden neuron and an output neuron. A quite good result on signal prediction has been achieved by this variant.[213]

In Ref. 107, a more general signal prediction algorithm is proposed for training RBF nets. In the consideration that in real time signal prediction the observations are presented sequentially and only once, they estimate a RBF net's parameters $\Theta = \{W, \mathbf{c}_j, \Sigma_j, j = 1, \ldots, n_h\}$ by minimizing the following modified error criterion instead of that given by Eq. (56);

$$\Theta(t) = \arg\min_\Theta \int_{\mathbf{x}\in D} ||F(\mathbf{x},\Theta) - F(\mathbf{x},\Theta(t-1))||^2 d\mathbf{x}, \tag{74}$$

subjected to the constraint $F(\mathbf{x}_t,\Theta) = \mathbf{d}_t$ or $(||F(\mathbf{x}_t,\Theta) - \mathbf{d}_t||^2 \le t_h$ and t_h is a prespecified error limit) where $\{\mathbf{x}_t, \mathbf{d}_t\}$ is a pair of training observation at the present time t, $F(\mathbf{x}_t,\Theta) = [f_1(\mathbf{x},\Theta), f_2(\mathbf{x},\Theta), \ldots, f_m(\mathbf{x},\Theta)]^t$ given by Eq. (67) or Eq. (69) and $D \subset R^n$ is the input pattern space.

Before closing this subsection, we would also like to note that RBF nets have close relations to *Parzen window* estimator for probability density[55] and the *probabilistic nets* proposed in Refs. 209 and 210. For the classification tasks, let the training samples $\mathbf{x}_j$'s be redenoted by $\mathbf{x}_{ij}, j = 1, \ldots, P_i, i = 1, \ldots, m$, where $\sum_{i=1}^m P_i = P$ and $\mathbf{x}_{ij}$ represents the jth sample of class i. The *Parzen window*

estimator is given by

$$f_i(\mathbf{x}) = \frac{1}{P_i\sigma_i}\sum_{j=1}^{P_i}\phi\left(\frac{||\mathbf{x}-\mathbf{x}_{ij}||}{\sigma_i}\right), \quad i = 1,\ldots,m, \tag{75}$$

similarly where $\phi(.)$ can be either Gaussian function $G(r) = \exp(-r^2)$ or a number of alternatives[209] and σ is chosen as a function of P_i such that $\sigma_i \to 0$, for $P_i \to \infty$ and $P_i\sigma_i \to \infty$, for $P_i \to \infty$. Here the differences from Eq. (67) are:

(1) all the $w_{ij} = 1$ for any i, j;
(2) the output f_i's are decoupled between the samples of different classes;
(3) σ_i is a function of number P_i of each class.

In fact, here each $f_i(\mathbf{x})$ is an estimation of the probability density of class i. In the *probabilistic nets*,[209,210] σ_i is fixed and can be the same for all the classes. When $m = 1$, the probabilistic net is just the special case of RBF net with fixed $w_{ij} = 1$. When $m > 1$, the probabilist nets can be regarded as m such special RBF nets with each of which having only one output variable.

4.2. *Other supervised learning models*

Let us further continue to look at the second and third categories mentioned in the beginning of Sec. 4 the second category consists of several other interesting supervised learning models. The third category consists of some models of complex structures using one or more the previously described learning models as components.

For the second category, we briefly introduce three interesting models as follows;

- *Kohonen learning vector quantization algorithms.* The algorithms are proposed for training a one layer net for classification or vector quantization. The layer consists of a number of units with weight vectors $\mathbf{w}_k, k = 1,\ldots,n_k$. One or several such weight vectors are used as the prototype vectors of each of m pattern class. Initially, all the weight vectors are set randomly or estimated by some unsupervised learning algorithm (e.g. Kohonen map[116]). Then they are modified by a training set of samples. For each sample x, a winner unit $k*$ is found by $k* = \arg_k||\mathbf{x}-\mathbf{w}_k||^2$ and the weight vector $\mathbf{w}_{k*}$ is modified according to the following rule

$$\begin{aligned}\Delta\mathbf{w}_{k*} &= \alpha(\mathbf{x}-\mathbf{w}_{k*}), && \text{if label}(\mathbf{w}_{k*}) = \text{label}(\mathbf{x})\\ &-\alpha(\mathbf{x}-\mathbf{w}_{k*}), && \text{otherwise},\end{aligned} \tag{76}$$

where $0 < \alpha < 1$ is a learning rate. Moreover, label $(\mathbf{x})$ denotes the class label of $\mathbf{x}$ and label $(\mathbf{w}_{k*})$ denotes the class that $\mathbf{w}_{k*}$ represents for. The algorithm is simply called LVQ.[116]

An improved version of this algorithm called LVQ2 (which is closer in effect to Bayes decision theory when used as classifier) has also proposed by Kohonen.[116] Let $k*$ again denotes winner unit and let k' denote the second winner (i.e. $k' = \arg_{k\neq k*}||\mathbf{x}-\mathbf{x}_k||^2$). The LVQ2 learning rule is given by

$$\begin{aligned}\Delta\mathbf{w}_{k*} &= -\alpha(\mathbf{x}-\mathbf{w}_{k*})\\ \Delta\mathbf{w}_{k'} &= \alpha(\mathbf{x}-\mathbf{w}_{k'}).\end{aligned} \tag{77}$$

The rule is not implemented for every sample $\mathbf{x}$ but only when label$(\mathbf{x}) \neq$ label$(\mathbf{w}_{k*})$, label$(\mathbf{x}) =$ label$(\mathbf{w}_{k'})$ and that $\mathbf{x}$ falls in a small window centered at the decision boundary (perpendicular bisector plane) between $\mathbf{w}_{k*}$ and $\mathbf{w}_{k'}$.

It was found that there are two places which could be further improved. First, because the updates are proportional to the difference of $\mathbf{x}$ and $\mathbf{w}_{k*}$ or $\mathbf{x}$ and $\mathbf{w}_{k'}$, the update on $\mathbf{w}_{k*}$ (correct class) is of larger magnitude than that on $\mathbf{w}_{k'}$ (wrong class); this results in monotonically decreasing distances $||\mathbf{w}_{k*}-\mathbf{w}_{k'}||$. Second, the continuation of learning from a good solution may lead to another asymptotic equilibrium of suboptimum solution. The two problems were solved by a new version called LVQ3[117]

$$\begin{aligned}\Delta\mathbf{w}_i &= -\alpha(\mathbf{x}-\mathbf{w}_i)\\ \Delta\mathbf{w}_j &= \alpha(\mathbf{x}-\mathbf{w}_j),\end{aligned} \tag{78}$$

where $\mathbf{w}_i, \mathbf{w}_j$ are two closest ones to $\mathbf{x}$. The rule is implemented when label$(\mathbf{w}_i) \neq$ label$(\mathbf{w}_j)$ and label$(\mathbf{x}) =$ label$(\mathbf{w}_j)$ as well as $\mathbf{x}$ falls in the above small window. In addition, when label$(\mathbf{w}_i) =$ label$(\mathbf{w}_j) =$ label$(\mathbf{x})$, Eq. (78) is replaced by

$$\begin{aligned}\Delta\mathbf{w}_i &= \alpha_1(\mathbf{x}-\mathbf{w}_i),\\ \Delta\mathbf{w}_j &= \alpha_1(\mathbf{x}-\mathbf{w}_j),\end{aligned} \tag{79}$$

where α_1 is a learning rate different from α.

- *Feedforward Boltzmann Machine.*

As mentioned in the introduction section, the Boltzmann Machine learning algorithm[2] is one of the two main driving forces which caused a new boost of the study on supervised learning neural nets. A Boltzmann machine consists of some fixed

number of two-valued units linked by symmetrical connections w_{ij}'s which forms a network sometimes called a Hopfield net. A set of specific values taken by all the unit is called a state of the net. An "energy" function over states is defined and it in turn induces a Boltzmann distribution over states in which low-energy states are more probable than high-energy states. During learning, each sample of the training set is a vector consisting of a number of binary values and each value clamps one unit. All the clamped units called "visible" units and the others called "hidden" units. The values that "hidden" units take are free but obey Boltzmann distribution defined by the energy function. The purpose of the learning algorithm given by Ref. 2 is to estimate the connection weights w_{ij}'s to maximize the log-likelihood over all the samples of the training set. Practically, the maximization is implemented by gradient-ascent method in cooperation with simulated annealing technique. Strictly speaking, Boltzmann machine is not a feedforward net but a recurrent network. Thus, we will not go into details about it. Here what we want to mention is that a similar learning procedure has been recently developed for feedforward nets (e.g. multilayer perceptron) with the same purpose as Boltzmann machine by replacing all the symmetrical connections with forward connections which resulted in what here we called *feedforward Boltzmann machine*.[155] For Boltzmann machine, the gradient of the log-likelihood function based on the symmetrical structure contains both a positive term and a negative term which makes Boltzmann learning consists of two phases — positive phase and negative phase respectively. The two phases are both indispensable for the learning to reach a stable balance. However, the negative phase have the disadvantages of increasing computation considerably and being more sensitive to statistical errors. Interestingly, for feedforward Boltzmann machine, the negative term is automatically eliminated in the gradient of the log-likelihood function based on the forward structure. This increases the learning speed of Boltzmann machine which is painfully slow. It was also found that these forward nets have close relation with *Belief networks*[167] developed in the literature of uncertainty reasoning in artificial intelligence.

- *Learning from examples and queries.*
  Up to now, all the learning algorithms which we have discussed are based on a set of training data $D_{tr} = \{(\mathbf{x}_p, \mathbf{d}_p), p = 1, \ldots, N_p\}$, where for each $\mathbf{x}_p$ there already exists a desired output $\mathbf{d}_p$ and each of such pair forms an example. In other words, all these algorithms learn from examples. Recently, several researchers attempt to introduce queries into some learning algorithms too (Refs. 19, 99 and 249). For such an algorithm, first some examples are used for learning and then the partially trained algorithm generates one or a number of inputs $\mathbf{x}_p, p = 1, \ldots, m_q$ and asks a supervisor to give their corresponding desired outputs. That is, in addition to using examples in a prespecified training set, many examples are deliberately generated, in response to queries and used during the learning process. This policy may give several advantages. First, for a real application of only a small set of training samples, the query provides a plausible way for collecting more examples. Second, some examples (e.g. those near decision boundaries when a net is used for classification purpose) are more critical than others and the query can pay more attention on obtaining these examples so that the required time and the number of examples can be reduced and the performance of the net can be improved.

  In Ref. 19, an algorithm is proposed to train, from examples and queries, a classification network with n input variables connected to k hidden threshold units. The learning process consists of two steps. First, k hyperplanes are searched for k hidden units as their separating hyperplanes. Then, the perceptron algorithm (or some other algorithm) is used for training the output layer. The key idea for finding a separating plane is as follows:

  Let $\mathbf{x}_+$ be a positive example and $\mathbf{x}_-$ be a negative example (e.g. $\mathbf{x}_+$ belongs to one class and x_- does not belong to the class). Let $\mathbf{x} = (\mathbf{x}_+ + \mathbf{x}_-)/2$ and query whether $\mathbf{x}$ is a positive or negative example. If positive, query the point halfway between $\mathbf{x}$ and $\mathbf{x}_-$ and so on. Repeating this process t times gives a point $\mathbf{y}_0$ on a separating hyperplane with t bits of precision. Then let $\mathbf{q} = \mathbf{y}_0 + \boldsymbol{\epsilon}$ for $\boldsymbol{\epsilon}$ a small random vector and query whether $\mathbf{q}$ is a positive or negative example. If (say) negative, query a point near $\mathbf{q}$ in the direction of x_+. A few queries again suffice to give a point $\mathbf{p}$ on a separating hyperplane. We may establish rapidly that $\mathbf{y}_0$ and $\mathbf{p}$ lie on the same separating hyperplane, since if they do, so do other points on the line which connects them. By repeating the process for several perturbations $\boldsymbol{\epsilon}$, we may find

n points on the same hyperplane which determine the hyperplane.

Based on this key idea, given a set of example pairs $S_p = \{(\mathbf{x}_+^i, \mathbf{x}_-^i), i = 1, \ldots, N\}$, a reasonable search process is to randomly call a pair from S_p without replacement and for each pair find a separating plane until either

(a) we have found k distinct separating planes, in which case we know we are done, or
(b) we have exhausted all pairs in S_p.

In Refs. 99 and 249, the query techniques are combined into the learning process of backpropagation for multilayer perceptron. First, a given training set is used to train a network and the partially trained net generates input samples which are located near decision boundaries through queries. Second, these queried examples are used in the same manner as using examples of those training sets to refine the network by backpropagation again. The key point for generating querying samples is inverse mapping, i.e. giving an output vector, one tries to find the corresponding input vector or vectors. In Ref. 99, an output vector $\mathbf{d}$ which reflects a class boundary (e.g. a vector has its qth component being 0.5 can reflect the boundary of the q class) is used as the desired output and then the corresponding input vector $\mathbf{x}$ is searched so that the least square errors E_2 is minimized. The search is implemented in such a way that the error E_2 is backpropagated down to the input level without changing weights of the nets and then the input is changed according to

$$\mathbf{x}(t+1) = \mathbf{x}(t) - \alpha \frac{\partial E_2}{\partial \mathbf{x}(t)} . \tag{80}$$

As a result, the trajectory of these changes forms a set of input examples which roughly constitutes a decision boundary. In sequence, for each point on the boundary, a conjugate pair which is near the boundary point and perpendicular to the boundary is chosen as a pair of querying samples. In this way, by inverting every vector which reflects the boundary of its correspondent class we can get a lot of querying samples near every decision boundary. A different way of generating querying samples is proposed in Ref. 249. There, a seed input $\mathbf{x}$ from one class is presented to a trained networks and a desired output $\mathbf{d}$ of another input from a different class is used as the desire output to form a pair $(\mathbf{x}, \mathbf{d})$. The error E_2 corresponding to the pair is backpropagated down to the input level without changing weights of the nets, and the input is changed according to Eq. (80). Then the points on the resulted trajectory of these changes are used as querying samples.

The third category consists of models of various complex structures, e.g. modular architecture, hierarchy architecture and others. These complex structures take some of the previously reviewed models as components so that the characteristics of different learning models can be combined, the computation costs can be reduced and the performances can be improved. In the following, we introduce several examples of this category:

- *Adaptive mixtures of modules.*
  Given subnets (or called modules/experts) $s_i, i = 1, \ldots, n_m$, with each having input $\mathbf{x}$ and output $\mathbf{y}_i$ and given another subnet called a gating network which has n_m output variables $g_i, i = 1, \ldots, n_m$ with $g_i > 0$ and $\sum_{i=1}^{n_m} g_i = 1$ (for a net with outputs $o_i, i = 1, \ldots, n_m$ which do not satisfy the condition, one can just normalize these outputs by $g_i = e^{o_i} / \sum_{i=1}^{n_m} e^{o_i}$), we mixture the outputs of subnets by weighted sum

$$\mathbf{y} = \sum_{i=1}^{n_m} g_i \mathbf{y}_i . \tag{81}$$

  Then we use a training set $D_{tr} = \{(\mathbf{x}_p, \mathbf{d}_p), p = 1, \ldots, N_p\}$ to modify the weights of each subnets as well as the gating net so that the following function is maximized by gradient ascent method

$$\ln L_p = \ln \sum_{i=1}^{n_m} g_i e^{-\frac{1}{2\sigma_i^2} \|\mathbf{y}(\mathbf{x}_p) - \mathbf{d}_p\|^2} , \tag{82}$$

  where σ_i^2 are scaling parameters associated with the ith subnet and these parameters are also modified by gradient ascent. Either the gating net or each of the subnets can be a multilayer feedforward net which is trained by chain rule to get the gradients with weights in the lower levels. This architecture was recently proposed by Refs. 103 and 159 and it has been reported that this architecture can decompose a task into several components and each of them is distributed to a subnet through competitive and associative learning which is naturally emerged by the maximization of Eq. (82) in gradient ascent way. In Ref. 244, a different version of the architecture was proposed for combining the classification results of multiple

classifiers. It has three main differences from that given in Refs. 103 and 159. First, each classifier (corresponding to each module here) can be either a neural net or any other conventional classifier. Second, each classifier (or equivalently all the weights of each subnet) have already been trained before considering combination and the purpose here is just to combine a number of already designed classifiers instead of training both the gating net and each subnet in the same time. Third, the parameters of the gating net are adjusted not by maximizing Eq. (82) but by a criterion based on an induced vector which accounts the correctness of each classifier for each input.

- *Hierarchical structures.*
  For the pattern classification problems, the tree classifier is quite useful when the number N_c of class is large and/or the dimension n of input $\mathbf{x}$ is high. In a tree classifier, a classifier at the root node divide a given training pattern set into at most N subsets. Each subset is assigned to a child node. This process is repeated for each child node in a recursive manner until each subset corresponds to a single class. These single class subsets correspond to the leaf nodes of the tree. The leaf nodes are labeled according to the class they represent. To design such a tree, there are several issues to be considered. First, at each node, given m child nodes, there are many ways to partition a subset into m smaller subsets. Moreover, m can also take a number of values (at most N_c values). Second, a subset can be divided based on either all of the component variables of $\mathbf{x}$ or just on a subset of the set of n components. This again gives a lot of possibilities. Third, for each node how can we design a suitable classifier to partition its subset? In the literature of conventional pattern recognition and statistics decision analysis, there have been a great number of studies on tree classifiers. The results of these studies can be directly adopted to neural network literature. The simple way is just let each node of a tree be a neural network classifier as what was made in Refs. 214 and 187. Due to the tree structure, each classifier can be very simple, e.g. in Ref. 187 each classifier is just a one layer feedforward network.
  Hierarchical structure can also be used for regression problem. The so-called *Bumptree* is proposed for the purpose.[160] By this model, a training data set is organized into a tree data structure with each node covering a bump-like subset of input pattern space. A bumplike subset is defined by a so-called bump function which specifies the range of support and the size of the bump. The size of the bump-like subset gradually decreases from the root to a leaf. At each leaf, an affine regression is made based on the training samples falling within the corresponding local bump region. When a test pattern comes, a searching process is made to find those leaf nodes that their bump regions over the input pattern, then the results of every regression function of these leaf nodes are weightedly summed to give the final regression value for the current input $\mathbf{x}$ (where the related weights are determined by the normalized values of the bump function at each of these leaf nodes).
- *Other structures.*
  There are also a number of other ways for combining different learning model to form a complex structure, e.g. in Ref. 23, a multilayer perceptron trained by backpropagation is used as a module at the first stage and then its output is further inputed to a module based Kohonen's LVQ1 learning algorithm. Some more examples are given in Refs. 40, 16 and 25.

5. Summary

Recent advances on techniques of static feedforward networks with supervised learning have been rather systematically reviewed. The survey was made by summarizing a great number of developments into four aspects:

(1) various improvements and variants made on the classical backpropagation techniques for Multilayer (static) perceptron nets.
(2) a number of other learning methods for training *Multilayer (static) perceptron.*
(3) various other feedforward models which are also able to implement function approximation, probability density estimation and classification.
(4) models with complex structures, e.g. modular architecture, hierarchy architecture and others.

These aspects basically cover the whole picture of the present state of supervised learning techniques for training static feedforward networks. We should note that the theoretical aspects of supervised learning for static feedforward networks have also experienced a tremendous development with many interesting theoretical results in the recent years.

However, due to space limitations, we have to leave the survey on this aspect elsewhere.

6. Acknowledgements

S. Klasa and L. Xu would like to acknowledge the support from Concordia University, Faculty Research Development Programme for Grant No. RO-20-I333. A. Yuille and L. Xu would like to acknowledge the support from DARPA with AFOSR-89-0506.

References

1. S. Abe, "Learning by parallel forward propagation," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 99–104.
2. D. H. Ackely, G. E. Hinton and T. J. Sejnowski, "A learning algorithm for Boltzman machine," *Cognitive Sci.* **9**, 147–169 (1985).
3. R. B. Allen and C. A. Kamm, "A recurrent neural networks for word identification from continuous phoneme strings," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 206–212.
4. L. G. Allre and G. E. Kelly, "Supervised learning techniques for backpropagation networks," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 721–728.
5. A. Amari, "Mathematical foundations of neurocomputing," *Proc. of IEEE* **78**, 1443–1463 (1990).
6. T. J. Anastasio, "A recurrent neural network model of velocity storage in the vestibule-ocular reflex," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 32–38.
7. D. Andes, B. Widrow, M. Lehr and E. Wan, "MRIII: A robust algorithm for training analog neural networks," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C, 1990) Vol. I, pp. 533–536.
8. G. K. Atkin, J. E. Bowcock and N. M. Queen, "Solution of a distributed deterministic parallel network using simulated annealing," *Pattern Recognition* **22** 461–466 (1989).
9. N. Baba, "A new approach for finding the global minimum of error function of neural networks," *Neural Networks* **2**, 367–374 (1989).
10. P. Baldi, "Computing with arrays of Bell-shaped and Sigmoid functions," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 735–742.
11. E. Barnard, "Performance and generalization of the classification figure of merit criterion function," *IEEE Trans. Neural Networks* **2**, 322–318 (1991).
12. A. Barron and R. Barron, "Statistical learning networks: A unifying view," in *Comput. Sci. and Statis. Proc. 20th Symp. Interface*, ed. E. Wegman, *Am. Statist. Assoc.* (Washington D.C., 1988) pp. 192–203.
13. A. Barron, "Statistical properties of artificial neural networks," in *Proc. IEEE Decision and Control* (Tampa, Florida, 1989) pp. 280–285.
14. B. G. Batchelor and B. R. Wilkins, "Adaptive discriminant functions," *IEE Conf. Publication, Vol. 42* (1968) pp. 168–178.
15. B. G. Batchelor, *Practical Approach to Pattern Recognition* (Plenum Press, New York, 1974).
16. R. Batruni, "A Multilayer neural network with piece-wise-linear structure and backpropagation learning," *IEEE Trans. Neural Networks* **2**, 395–403 (1991).
17. E. B. Baum and F. Wilczek, "Supervised learning of probability distributions by neural networks," in *Neural Information Processing Systems*, ed. D. Z. Anderson (New York, 1988) pp. 53–61.
18. E. B. Baum and K. J. Lang, "Constructing hidden units using examples and queries," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 904–910.
19. E. B. Baum, "Neural net algorithms that learn in polynomial time from examples and queries," *IEEE Trans. Neural Networks* **2**, 5–19 (1991).
20. R. Beale and T. Jackson, *Neural Computing* (Adam Hilger, 1990).
21. S. Becker and Y. Le Cun, "Improving the convergence of backpropagation learning with second order methods," in *Proc. 1988 Connectionist Models Summer School, Pittsburg 1988*, eds. D. Touretzky, G. Hinton and T. Sejnowski (Morgan Kaufmann, San Mateo, 1988) pp. 29–37.
22. U. Bodenhausen, "Learning internal representation of pattern sequences in a neural network with adaptive time-delays," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 113–118.
23. M. de Bollivier, P. Gallinari and S. Thiria, "Cooperation of neural nets for robust classification," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. II, pp. 113–120.
24. S. M. Botros and C. G. Atkeson, "Generalization properties of radial basis function," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 707–713.
25. L. Bottou and P. Gallinari, "A framework for the cooperation of learning algorithms," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 781–788.
26. L Breman, J. H. Fredman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees* (Wadsworth International, CA: Belmont, 1984).
27. R. P. Brentt, "Fast training algorithms for multilayer neural nets," *IEEE Trans. Neural Networks* **2**, 346–354 (1991).

28. D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Compl. Syst* **2**, 321–323 (1988).
29. A. E. Bryson and Y. C. Ho, *Applied Optimal Control* (Blaisdell, New York, 1969).
30. P. Burrascano and P. Lucci, "Smoothing backpropagation cost function by delta constraining," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 75–78.
31. P. Burrascano, "A norm selection criterion for the generalized delta rule," *IEEE Trans. Neural Networks* **2**, 125–130 (1991).
32. G. A. Carpenter, "Neural network models for pattern recognition and associative memory," *Neural Networks* **2**, 243–257 (1989).
33. L. W. Chan and F. Fallside, "An adaptive training algorithm for backpropagation networks," *Comput. Speech and Language* **2**, 205–218 (1987).
34. Y. Chauvin, "Generalization performance of overtrained backpropagation," in *Proc. Eurasip Workshop on Neural Networks*, eds. L. B. Almedia and C. J. Wellekens, (Springer-Verlag, 1990) pp. 46–55.
35. Y. Chauvin, "A backpropagation algorithm with optimal use of hidden units," in *Advances in Neural Information Processing Systems*, ed. D. Touretzky, (Morgan Kaufmann, 1989).
36. T. R. Chay, "Complex oscillations and chaos in a simple neural model," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 657–662.
37. M. Chen and M. T. Manry, "Backpropagation representation theorem using power series," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, 643–648.
38. J. R. Chen and P. Mars, "Stepsize variation methods for accelerating the backpropagation algorithm," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C, 1990) Vol. I, pp. 601–604.
39. S. Chen, C. F. N. Cowan and P. M. Grant, "Orthogonal least squares learning algorithm for Radial basis function networks," *IEEE Trans. Neural Networks* **2**, 302–309 (1991).
40. S. B. Cho and J. H. Kim, "Hierarchically structured neural networks for printed hangu; character recognition," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 265–270.
41. D. Cohen and J. Shawe-Taylor, "Feedforward networks — a tutorial," in *New developments in Neural Computing*, eds. J. G. Taylor and C. L. T. Mannion (Adam Hilger, 1990).
42. P. W. Cooper, "The hypersphere in pattern recognition," *Information and Control* **5** (1962).
43. P. W. Cooper, "A note on an adaptive hypersphere decision boundary," *IEEE Trans. Elect. Comp.* 984–959 (1966).
44. T. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Elect. Comput.* **EC – 14**, 326–334 (1965).
45. J. D. Cowan and D. H. Sharp, "Neural nets and artificial intelligence," in *Proc. of the Am. Acad. of Arts and Sci.* **117**, 85–121 (1988).
46. J. D. Cowan and D. H. Sharp, "Neural nets," *Quart. Rev. of Biophys.* **21**, 365–427 (1988).
47. S. P. Day and D. S. Camporese, "Continuous-time temporal backpropagation," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 95–100.
48. J. Dayhoff, *Neural Network Architectures: An Introduction* (Van Nostrand Reinhold, 1989).
49. S. P. Day and D. Camporese, "A stochastic training technique for feedforward neural networks," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 607–612.
50. A. Dembo and T. Kailath, "Model free distributed learning," *IEEE Trans. Neural Networks* **1**, 58–70 (1990).
51. P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach* (Prentice Hall, 1982).
52. B. W. Dickinson, "Group behavior models for learning in neural networks," in *Proc. IEEE Decision and Control* (Tampa, Florida, 1989) Vol. I, pp. 249–251.
53. S. C. Douglas and T. H. Meng, "Linearized least square training of multilayer feedforward neural networks," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 307–312.
54. H. Drucker and Y. LeCun, "Double backpropagation increasing generalization performance," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 145–150.
55. R. O. Duda and P. E. Hart, *Pattern Classification And Scene Analysis* (Wiley, 1973, New York).
56. J. L. Elman, "Finding structure in time," *Cognitive Sci* **14**, 79–211 (1990).
57. El-Jaroudi and J. Makhoul, "A new error criterion for posterior probability estimation with neural nets," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 185–192.
58. S. E. Fahlman, "Faster-learning variations on backpropagation: An empirical study," in *Proc. 1988 Connectionist Models Summer School (Pittsburg, 1988)*, eds. D. Touretzky, G. Hinton and T. Sejnowski (Morgan Kaufmann, San Mateo, 1988).
59. S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," *Advances in Neural Information Processing Systems II, Denver, 1989* ed. D. Touretzky (Morgan Kaufmann, 1990) pp. 524–532.
60. W. Fakhr and M. I. Elmasry, "A fast learning technique for the multilayer perceptron," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 257–262.
61. D. Figueiredo, "An optimal matching-score net for pattern classification," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 906–909.
62. R. Franke, "Scattered data interpolation: Tests of some methods," *Math. Comp.* **38(5)**, 181–200.
63. M. Frean, "The upstart algorithm: A method for constructing and training feedforward neural networks," *Neural Comput.* **2**, 198–209 (1990).
64. J. H. Friedman and W. Stuetzle, "Projection Pur-

suit regression," *J. Am. Statis. Assoc.* **76 : 367**, 817–8243 (1981).
65. J. H. Friedman, "Adaptive spline networks," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991). pp. 675–683.
66. S. Gallent, "A connectionist learning algorithm with provable generalization and scaling bounds," *Neural Networks* **3**, 191–201 (1990).
67. T. Geszti, *Physical Models of Neural Networks*, (World Scientific, Singapore, 1990).
68. F. Girosi and T. Poggio, "Networks and the best approximation property," M.I.T. AI Memo. No. 1164, MIT (1989).
69. F. Girosi and T. Poggio, "Extensions of a theory of networks for approximation and learning: Outlier and negative examples," M.I.T. AI Memo. No. 1220, MIT, (1990); also in *Advances in Neural Information Processing System 3*, eds. (R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 750–756.
70. C. L. Giles and T. Maxwell, "Learning, invariance and generalization in high-order neural networks," *Appl. Optics* **26 : 23**, 4985–4992 (1987).
71. H. Gish, "A probabilist approach to the understanding and training of neural network classifiers," in *Proc. of IEEE Conf. on ASSP* (1990) pp. 1361–1364.
72. S. D. Goggin, K. E. Gustafson and K. M. Johnson, "An asymptotic singular value decomposition analysis of nonlinear multilayer neural networks," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 1785–789.
73. D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, 1989).
74. S. Grossberg, *Adaptive Brain*, 2 Vols (Elsevier Amsterdam, 1987).
75. S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Sci.* **11**, 23–63 (1987).
76. M. Gutierrez 89, J. Wang and R. Grondin, "Estimating hidden unit number for two-layer perceptrons," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 677–681.
77. M. Hagiwara, "Accelerated backpropagation using unlearning based on Hebb rule," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C, 1990) Vol. I, pp. 617–620.
78. M. Hagiwara, "Novel backpropagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection" in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 625–630.
79. J. B. Hampshire and A. H. Waibel, "A novel objective function for improved phoneme recognition using time-delay neural networks," *IEEE Trans. on Neural Networks* **1**, 216–228 (1990).
80. S. J. Hanson and L. Y. Pratt, "Some comparisons of constraints for minimal network construction with backpropagation," *Advances in Neural Information Processing Systems*, ed. D. Touretzky (Morgan Kaufmann, 1989).
81. S. J. Hanson and D. J. Burr, "Minkowski-r backpropagation: Learning in connectionist models with non-Euclidian error signals," *Neural Information Processing Systems*, ed. D. Z. Anderson (New York, 1988) pp. 349–357.
82. R. F. Harrison, S. J. Marshall and R. L. Kennedy, "The early diagnosis of heart attacks: A neurocomputational approach," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 1–6.
83. R. Hecht-Nielsen, *Neurocomputing* (Addison-Wesley, 1991).
84. J. Hertz, J. Krogh and R. G. Palmer, *Introduction to the Theory of Neural Computation* (Addison-Wesley, 1991).
85. Y. Hirose, K. Yamashita and S. Hijiya, "Backpropagation algorithm which varies the number of hidden units," *Neural Networks* **4**, 61–66 (1991).
86. G. E. Hinton, "Learning translation invariant recognition in a massively parallel networks," *Lect. Notes in Comput. Sci.* **258**, 1–13, (1987).
87. G. E. Hinton, "Connectionist learning procedures," *Artif. Intell* **40**, 185–234 (1989).
88. Y. C. Ho and R. L. Kashyap, "An algorithm for linear inequalities and its applications," *IEEE Trans. on Elect. Comput.* **EC – 14**, 683–688 (1965).
89. M. Holt and S. Semnani, "Convergence of backpropagation in neural networks using a log-likelihood cost function," *Elec. Lett.* **26**, 1964–1965 (1990).
90. T. L. Holt and T. E. Baker, "Backpropagation simulations using limited precision," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 121–126.
91. K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks* **4**, 251–257 (1991).
92. K. Hornik, S. Stinchocombe and H. White, "Multilayer feedforward networks are universal approximators, derivatives using multilayer," *Neural Networks* **2**, 359–366 (1989).
93. K. Hornik, S. Stinchocombe and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks* **3**, 551–560 (1990).
94. J. C. Hoskins, "Speeding up artificial neural networks in the real world," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1989) Vol. II, pp. 626.
95. P. J. Huber, "Projection pursuit," *Annal. Statis.* **13 : 2**, 435–475 (1985).
96. W. Huang and R. Lippmann, "Comparisons between neural network and conventional classifiers," in *Proc. IEEE Int. Conf. on Neural Networks*, (San Diego, CA. 1987) Vol. IV, pp. 485–493.
97. Q. Huang *et al.*, "Identification of firing patterns of neuronal signal," in *Proc. of IEEE Decision and Control 1989. Tampa, Florida* (1989) Vol. I, pp. 266–271.
98. M. J. Hudak, "RCE networks: An experimental

investigation," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 849–854.
99. J. H. Hwang, J. J. Choi, S. Oh and R. J. Marks II, "Query-based learning applied to partially trained multilayer perceptrons," *IEEE Trans. on Neural Network* **2**, 131–136 (1991).
100. C. Ioss, "From lattices of phonemes to sentences: A recurrent neural network approach," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 833–838.
101. M. Ishikawa, "A structural learning algorithm with forgetting of weight link weights," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1989) Vol. II.
102. R. A. Jacobs, "Increased rate of convergence through learning rate adaptation," *Neural Networks* **1**, 295–307 (1989).
103. R. A. Jacobs and Michael I. Jordan, "A competitive modular connectionist architecture," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 767–773.
104. A. K. Jain, "Advances in statistical pattern recognition," in *Pattern Recognition Theory and Applications*, eds. P. A. Devijver and J. Kittler (Springer-Verlag, New York, 1989).
105. R. D. Jones *et al.*, "Information theoretic derivation of network architecture and learning algorithms," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 473–478.
106. M. Jordan, "Generic constraints on underspecified target trajectories," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1989) Vol. I, pp. 217–225.
107. V. Kardirkamanathan, M. Niranjan and F. Fallside, "Sequential adaptation of radial basis function neural networks and its application to time-series prediction," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 721–727.
108. B. L. Kalman and S. C. Kwasny, "A superior error function for training neural networks," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 49–52.
109. F. Kanaya and S. Miyake, "Bayes statistical behavior and valid generalization of pattern classifying neural networks," *IEEE Trans. Neural Networks* **2**, 471–475 (1991).
110. E. D. Karnin, "A simple procedure for pruning backpropagation trained neural networks," *IEEE Trans. Neural Networks* **1**, 239–242 (1990).
111. T. Khanna, *Foundations of Neural Networks* (Addison-Wesley, 1990).
112. M. S. Kim and C. C. Guest, "Modification of backpropagation network for complex-value signal processing in frequency domain," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 27–32.
113. S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by simulated annealing," *Science* **220**, 671–680 (1983).
114. S. Knerr, L. Personnaz and G. Dreyfus, "A new approach to the design of neural network classifiers and its application to the automatic recognition of handwritten digits," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 91–96.
115. T. Kohonen, "An introduction to neural computing," *Neural Networks* **1**, 3–16 (1988).
116. T. Kohonen, *Self-organization and Associative Memory* (3rd ed.) (Springer-Verlag, Berlin, 1989).
117. T. Kohonen, "The self-organizing map," *Proc. of IEEE* **78**, 1464–1480 (1990).
118. S. Kollias and D. Anastassiou, "An adaptive least square algorithms for efficient training of artificial neural networks," *IEEE Trans. Circuits and Syst.* **36**, 1092–1101 (1989).
119. J. I. Kolen and J. B. Pollack, "Backpropagation is sensitive to initial conditions," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 860–867.
120. J. R. Koza and J. P. Rice, "Genetic generation of both the weights and architecture for a neural networks," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 397–404.
121. M. A. Kraaijveld and R. P. W. Duin, "Generalization capabilities of minimal kernel-based networks," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 843–848.
122. A. H. Kramer and A. Sangiovanni-Vincentelli, "Efficient parallel learning algorithms for neural networks," in *Advances in Neural Information Processing Systems*, ed. D. Touretzky (Morgan Kaufmann, 1989) pp. 40–48.
123. V. Ya. Kreinovich, "Arbitrary nonlinearity is sufficient to represent all function by neural networks: A Theorem," *Neural Networks* **4**, 381–383 (1991).
124. J. K. Kruschke, "Improving generalization in backpropagation networks with distributed bottlenecks," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 443–447.
125. A. Krzyzak, W. Dai and C. Suen, "Classification of large set of handwritten characters using modified backpropagation model," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 225–232.
126. S. Y. Kung and Y. H. Hu, "A Frobenius approximation reduction method (FARM) for determining optimal number of hidden units," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 163–168.
127. Y. Lacouture, "Mean-variance backpropagation: A connectionist learning algorithm with a selective attention mechanism," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 31–36.
128. S. H. Lane *et al.*, "Multilayer perceptrons with B-spline receptive field functions," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 684–692.
129. K. J. Lang, A. H. Waibel and G. E. Hinton, "A time-

delay neural network architecture for isolated word recognition," *Neural Networks* **3**, 23–43 (1990).
130. Y. LeCun, "Generalization and network design strategies," Department of Computer Science, University of Toronto, TR CRG-TR-89-4 (1989).
131. Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.* **1**, 541–551 (1990).
132. S. Lee and R. M. Kil, "Robot kinematic control based on bidirectional mapping neural network," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 327–335.
133. Y. Lee and R. P. Lippmann, "Practical characteristics of neural network and conventional pattern classifiers on artificial and speech problems," *Advances in Neural Information Processing System II, Denver, 1989* (Morgan Kaufmann, 1990) pp. 168-177.
134. S. Lee and R. M. Kil, "A Gaussian potential function network with hierarchically self-organizing learning," *Neural Networks* **4**, 207–224 (1991).
135. Y. Lee, S. Oh and M. W. Kim, "The effect of initial weights on premature saturation in back propagation learning," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 765–770.
136. W. T. Lee, "On optimal adaptive classifier design criterion," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 385–390.
137. E. Levin, R. Gewirtzman and G. F. Inbar, "Neural network architecture for adaptive system modeling and control," *Neural Networks* **4**, 185–191 (1991).
138. S. Li, "An optimized backpropagation with minimium norm weights," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 697–702.
139. R. P. Lippmann, "An introduction to computing with neural sets," *IEEE ASSP Mag.* (1987) pp. 4–22.
140. H. Van der Maas, P. Verschure and P. Molenaar, "A note on chaotic behavior in simple neural networks," *Neural Networks* **3**, 119–122 (1990).
141. S. Makram-Ebeid, J. -A. Sirat and J. -R. Viala, "A rationalized backpropagation learning algorithm," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1989) Vol. II, pp. 373–380.
142. S. G. Mallat, "Multifrequency channel decomposition of images and wavelet models," *IEEE Trans. ASSP* **37 : 12**, 2091–2110 (1989).
143. J. M. Mcinerny, K. G. Haines, S. Biafore and R. Hecht-Nielsen, "Backpropagation error surfaces can have local minima" in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1989) Vol. II.
144. B. W. Mel and S. M. Omohundro, "How receptive field parameters affect neural learning," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 757–763.
145. M. Mezrd and J. -P. Nadal, "Learning in feedforward layered networks: The tiling algorithm," *J. Phys.* **A22**, 2191–2204 (1989).
146. G. Miller, P. Todd and S. Hedge, "Designing neural networks using genetic algorithms," in *Conf. on Genetic Algor. Proc. 3rd*, (Arlington, 1989).
147. A. A. Minai and R. D. Williams, "Backpropagation heuristics: A study of the extended delta-bar-delta algorithm," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 595–600.
148. M. L. Minsky and S. A. Papert, *Perceptrons* (MIT Press, Cambridge, 1988).
149. D. J. Montana and L. Davis, "Training feedforward networks using genetic algorithms," in *Proc. of 11th IJCAI* (Detroit, 1989) pp. 762–767.
150. J. Moody and J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Comput.* 1 281–294 (1989).
151. F. Mosteller and J. Tukey, *Robust Estimation Procedures* (Addison Wesley, 1980).
152. J. Movellan, "Error functions to improve noise resistance and generalization in backpropagation networks," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1990) Vol. I, pp. 557–560.
153. M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Advances in Neural Information Processing Systems*, ed. D. Touretzky, (Morgan Kaufmann, 1989) pp. 349–357.
154. K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks* **1**, 3–27 (1990).
155. R. M. Neal, "Learning stochastic feedforward networks," Technical Report, Connectionist research group at the University of Toronto, Canada (1990).
156. M. Nelson and W. T. Illingworth, *A Practical Guide to Neural Nets* (Addison-Wesley, 1991).
157. D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 21–26.
158. S. J. Nowlan, "Max likelihood competition in RBF networks," Technical Report CRG-Tr-90-2, Department of Computer Science University of Toronto (1990).
159. S. J. Nowlan and G. E. Hinton, "Evaluation of adaptive mixtures of competing experts," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 774–780.
160. S. M. Omohundro, "Bumptress for efficient function, constraint and classification learning," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 693–699.
161. Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks* (Addison-Wesley, 1989).
162. D. B. Parker, "Learning logic," Technicial Report, Center for Computational Search in Economics and Management Science, MIT (1985).

163. P. B. Parker, "Optimal algorithm for adaptive networks: Second order direct back propagation and second order hebbian learning," in *Proc. IEEE Int. Conf. on Neural Networks* (San Diego, CA. 1987) Vol. II, pp. 593–600.
164. Y. C. Pati and P. S. Krishnaprasad, "Discrete affine wavelet transforms for analysis and synthesis of feedforward neural networks," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 743–749.
165. A. Patrikar and J. Provence, "Learning by local variations," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1990) Vol. I, pp. 700–703.
166. J. C. Platt, "Learning by combining memorization and gradient descent," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 714–720.
167. J. Pearl, *Probabilistic Reasoning in Intelligent System: Networks of Plausible Inference* (Morgan Kaufmann, San Mateo, California, 1988).
168. B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Comput.* **1**, 263–269 (1989).
169. T. Poggio and F. Girosi, "A theory of networks for approximation and learning," M.I.T. AI Memo. No. 1140, MIT (1989).
170. T. Poggio and F. Girosi, "Networks for approximation and learning," in *Proc. of IEEE* **78**, pp. 1481–1497 (1990).
171. M. J. D. Powell, "Radial basis functions for multivariable interpolation: Review," in *Algorithms For Approximation*, eds. J. C. Mason and M. G. Cox (Clarendon Press, Oxford, 1987).
172. G. V. Puskorius and L. A. Feldkamp, "Decoupled extended Kalman filter training of feedforward layered networks," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 771–777.
173. S. Qian *et al.*, "Function approximation with an orthogonal basis net," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 605–619.
174. J. R. Quinlan, "Induction of decision trees," *Machine Learning* **1**, 81–106 (1986).
175. D. L. Reilly, L. N. Cooper and C. Elbaum, "A neural model for category learning," *Biol. Cybn.* **45**, 35–41 (1982).
176. D. L. Reilly, C. Scofield, C. Elbaum and L. N. Cooper, "Learning system architectures composed of multiple learning modules," in *Proc. IEEE Int. Conf. on Neural Networks* (San Diego, CA. 1987) Vol. II, pp. 495–503.
177. S. Renals and R. Rohwer, "Phoneme classification experiments using radial basis functions," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1989) Vol. I, pp. 462–467.
178. A. Rezgui and N. Tepedelenlioglu, "The effect of the slope of the activation function on the back propagation algorithm," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1990) Vol. I, pp. 707–780.
179. L. P. Ricotti, S. Ragazzini and G. Martinelli, "Learning word stress in a suboptimal second order backpropagation neural network," in *Proc. IEEE Int. Conf. on Neural Networks* (San Diego, CA 1988) Vol. I, pp. 355–361.
180. A. K. Rigler, J. M. Irvine and T. P. Vogl, "Rescaling of variables in backpropagation learning," *Neural Networks* **4**, 225–229 (1991).
181. F. Ronsenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.* **65**, 386–408 (1958).
182. F. Ronsenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* (Spartan Books, Washinton D.C., 1962).
183. D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing, Vol. 1* (MIT Press, 1986).
184. D. W. Ruck *et al.*, "The multilayer perceptron as an approximation to a Bayes optimal discriminant function," *IEEE Trans. Neural Networks* **1**, 296–299 (1990).
185. D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microbreak structure of Cognitive Processing, Vol. I & II* (MIT Press, 1986).
186. A. Saha *et al.*, "Oriented nonradial basis functions for image coding and analysis," *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 728–734.
187. A. Sankar and R. J. Mammone, "Optimal pruning of neural tree networks for improved generalization," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 219–224.
188. T. Sanger, "Basis-function trees as a generalization of local variable selection methods for function approximation," in *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 700-706.
189. T. Samad, "Backpropagation improvements based on heuristic arguments," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1990) Vol. I, pp. 565–568.
190. R. Scalero and N. Tepedelenlioglu, "A fast algorithm for neural network," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1990) Vol. I, 715–718.
191. D. V. Schriebman and E. M. Norris, "Speeding up backpropagation by gradient correlations," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1990) Vol. I, pp. 723–726.
192. C. L. Scofield, E. L. Reilly, C. Elbaum and L. N. Cooper, "Pattern class degeneracy in an unrestricted storage density memory," in *Neural Information Processing Systems*, ed. D. Z. Anderson (New York, 1988) pp. 674–682.
193. C. L. Scofield and D. L. Reilly, "Into silicon: Real

time learning in a high density RBF neural network," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 551–556.
194. I. Sethi, "Entropy nets: From decision trees to neural networks," in *Proc. of IEEE 78* (1990) pp. 1605–1613.
195. I. Sethi and M. Otten, "Comparison between entropy net and decision tree classifiers," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 63–68.
196. S. Shah and F. Palmieri, "MEKA-a fast local algorithm for training feedforward neural networks," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 41–46.
197. J. F. Shepanski, "Fast learning in artificial neural systems: Multilayer perceptron training using optimal estimation," in *Proc. IEEE Int. Conf. on Neural Networks* (San Diego, CA 1988) Vol. I, pp. 465–472.
198. J. S. Shawe-Talor, "Linear programming algorithm for neural networks," *Neural Networks* **3**, 575–582 (1990).
199. Y. Shin and J. Ghosh, "The Pi-Sigma network: An efficient high order neural network for pattern classification and function approximation," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 13–18.
200. P. A. Shoemaker, M. J. Carlin and R. L. Shimabukuro, "Backpropagation learning with trinary quantization of weight updates," *Neural Networks* **4**, 231–241 (1991).
201. P. A. Shoemaker, "A note on least-square learning procedures and classification by neural network models," *IEEE Trans. Neural Networks* **2**, 158–160 (1991).
202. J. Sietsma and R. J. F. Dow, "Neural net pruning — why and how," in *Proc. IEEE Int. Conf. on Neural Networks* (San Diego, CA 1988) Vol. I, pp. 333–335.
203. J. Sietsma and R. J. F. Dow, "Creating artifical neural networks that generalize," *Neural Networks* **4**, 67–79 (1991).
204. F. M. Silva and L. B. Almeida, "Acceleration techniques for the backpropagation algorithm," in *Lecture Notes In Computer Sci., 412* (1990) pp. 110–119.
205. S. Singhal and L. Wu, "Training Feedforward networks with the extended Kalman filter," in *Advances in Neural Information Processing Systems*, ed. D. Touretzky (Morgan Kaufmann, 1989) pp. 133–140.
206. S. A. Solla, E. Levin and M. Fleisher, "Accelerated learning in layered neural networks," *Compl. Syst.* **2**, 625–639 (1988).
207. F. J. Solis and J. B. Wets, "Minimization by random search techniques," *Math. of Operations Res.* **6**, 19–30 (1982).
208. J. Song and M. H. Hassoun, "Learning with hidden targets," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 93–98.
209. D. F. Specht, "Probabilistic neural networks," *Neural Networks* **3**, 109–118 (1990).
210. D. F. Specht, "Probabilistic neural networks and the polynomial Adaline as complementary techniques for classification," *IEEE Trans. Neural Networks* **1**, 111–121 (1990).
211. M. Stinchcombe and H. White, "Universal approximation using feedforward networks with nonsigmoid hidden layer activation functions," in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C., 1989) Vol. I, pp. 613–617.
212. M. Stinchcombe and H. White, "Approximation and learning unknown mappings using multilayer feedforward networks with bound weights," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. III, pp. 7–15.
213. K. Stokbro, D. K. Umberger and J. A. Hertz, "Exploiting neurons with localized receptive fields to learn chaos," preprint 90/28 S, Nordita, Copenhagen, Denmark (1990).
214. G. Z, Sun, Y. C. Lee and H. H. Chen, "A novel net that learns sequential decision process," in *Neural Information Processing Systems*, ed. D. Z. Anderson. (New York, 1988) pp. 760–766.
215. N. Tishby, E. Levin and S. Solla, "Consistent inference of probabilities in layered networks: Predictions and generalization," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1989) Vol. II, pp. 403–409.
216. T. Tollenaere, "SuperSAB: Fast adaptive back propagation with good scaling properties," *Neural Networks* **3**, 561–573 (1990).
217. F. S. Tsung, G. W. Cottrell and A. I. Selverston, "Some experiments on learning stable network oscillations," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 169–174.
218. P. E. Utgoff, "Perceptron trees: A case study in hybrid concept representations," *Connect. Sci.* **1**, 377–391 (1989).
219. T. P. Vogl *et al.*, "Accelerating the convergence of the backpropagation method," *Biol. Cybern.* **59**, 257–263 (1988).
220. A. Von Lehman *et al.*, "Factors influencing learning by backpropagation," in *Proc. IEEE Int. Conf. on Neural Networks* (San Diego, CA 1988) Vol. I, pp. 335–341.
221. Y. Wada and M. Kawato, "Estimation of generalization capability by combination of new information criterion and cross validation," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 1–6.
222. E. A. Wan, "Temporal back for FIR neural nets," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 575–580.
223. E. A. Wan, "Neural Network classification: A Bayesian interpretation," *IEEE Trans. Neural Networks* **1**, 303–305 (1990).
224. X. Wang, "Period-doublings to chaos in a simple neural networks," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 333–339.
225. R. L. Watrous, "Learning algorithms for connectionist networks: Applied gradient methods of

nonlinear optimization," in *Proc. IEEE Int. Conf. on Neural Networks* (San Diego, CA. 1987) Vol. II, pp. 619–627.
226. P. D. Wasserman, *Neural Computing: Theory and Practice* (Van Nostrand Reinhold, 1989).
227. A. S. Weigend, D. E. Rumelhart and B. A. Huberman, "Generalization by weight-elimination applied to currency exchange rate prediction," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. I, pp. 837–841; *Advances in Neural Information Processing System 3*, eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky (Morgan Kaufmann, San Mateo, 1991) pp. 875–882.
228. M. K. Weir, "A method for self-determination of adaptive learning rates in back propagation," *Neural Networks* **4**, 371–379 (1991).
229. P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral science," Ph. D. Thesis, Harvard University (1974).
230. P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks* **1**, 339–356 (1988).
231. P. J. Werbos, "Neural Networks for control and system identification," in *Proc. of the 28th Conf. on Decision and Control* (1989) pp. 260–265.
232. P. J. Werbos, "Backpropagation and neural control: A review and prospectus," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. I, pp. 209–216.
233. N. Weymaere and J. Martens, "A fast robust learning algorithm for feedforward neural networks," *Neural Networks* **4**, 361–369 (1991).
234. H. White, "Learning in artificial neural networks: A statistical perspective," *Neural Comput.* **1**, 425–464 (1989).
235. H. White, "An additional hidden unit test for neglected nonlinearity in multilayer feedforward networks," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. II, pp. 451–455.
236. D. Whitley and G. Hanson, "Optimizing neural networks using faster, more accurate genetic search," in *Proc. 3rd Conf. on Genetic Algor.* (Arlington, 1989) pp. 391–396.
237. D. Whitely and C. Bogart, "The evolution of connectivity: Pruning neural networks using genetic algorithm" in *Proc. Int. Joint Conf. on Neural Networks* (Washington D.C, 1990) Vol. I, pp. 134–137.
238. A. P. Wieland, "Evolving neural network controllers for unstable system," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 667–673.
239. B. Widrow, "Generalization and information storage in networks of Adaline 'neuron'," in *Self-Organizing Systems 1962*, eds. M. Yovitz, G. Jacobi and G. Goldstein (Spartan Books, Washington D. C., 1962).
240. B. Widrow, "30 years of adaptive neural networks: Perceptron, madaline and backpropagation," in *Proc. of IEEE 78* (1990) pp. 1415–1442.
241. R. J. Williams and D. Zipser, "A learning algorithm for continuously running fully recurrent neural networks," *Neural Comput.* **1**, 270–280 (1989).
242. L. Xu, "Adding learned expectation into the learning procedure of self-organizing maps," *Int. J. Neural Syst.* **1**, 269–283 (1990).
243. L. Xu and K. Gao, "A single neural unit can beat XOR problem: A neural net consisting of units with a new type of activation function," in *Proc. Int. Joint. Conf. on Neural Networks* (Beijing, 1992).
244. L. Xu, A. Krzyzak and C. Suen, "Associative switch for combining multiple classifers," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991). Vol. I, pp. 43–48.
245. L. Xu, A. Krzyzak and E. Oja, "A neural net for dual subspace pattern recognition methods," *Int. J. of Neural Syst.* **2**, 169–184 (1991).
246. L. Xu, E. Oja and C. Y. Suen, "Modified Hebbian learning for curve and surface fitting," *Neural Networks* **5**, 441–457 (1992).
247. L. Xu, Adam Krzyzak and E. Oja, "Rival penalized competitive learning for clustering analysis, RBF net and curve detection," to appear in *IEEE Trans. Neural Networks* (1991).
248. K. Yamada *et al.*, "Handwritten numeral recognition by multilayered neural network with improved learning algorithm," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990) Vol. II, pp. 259–266.
249. K. Yamada, "Multifont alpha-numeric recognition using multilayer neural networks with a rejection function," in *Proc. of Vision Interface'91* (1991).
250. K. Yamada, "Learning of category boundaries based on inverse recall by multilayer neural network," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991) Vol. II, pp. 7–12.
251. Y. Yao and W. J. Freeman,, "Model of biological pattern recognition with spatially chaotic dynamics," *Neural Networks* **3**, 153–170 (1990).
252. Y. H. Yu, "Descending epsilon in backpropagation: A technique for better generalization," in *Proc. Int. Joint Conf. on Neural Networks* (San Diego, 1990). Vol. III, pp. 167–172.
253. A. L. Yuille and N. M. Grzywacz, "A mathematical analysis of the motion coherence theory," *Int. J. of Computer Vision* **3**, 155–175 (1989).
254. Y. Zhao, "Projection pursuit learning," in *Proc. Int. Joint Conf. on Neural Networks* (Seattle, 1991). Vol. I, pp. 869–874.