Constructive Feedforward Neural Networks for Regression Problems: A Survey

Tin-Yau Kwok & Dit-Yan Yeung*

Technical Report HKUST-CS95-43
September 1995

* Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon
Hong Kong
{jamesk,dyyeung}@cs.ust.hk

Abstract

In this paper, we review the procedures for constructing feedforward neural networks in regression problems. While standard back-propagation performs gradient descent only in the weight space of a network with fixed topology, constructive procedures start with a small network and then grow additional hidden units and weights until a satisfactory solution is found. The constructive procedures are categorized according to the resultant network architecture and the learning algorithm for the network weights.

The Hong Kong University of Science & Technology
Technical Report Series
Department of Computer Science

1 Introduction

In recent years, many neural network models have been proposed for pattern classification, function approximation and regression problems. Among them, the class of multi-layer feed-forward networks is perhaps the most popular. Standard back-propagation performs gradient descent only in the weight space of a network with fixed topology; this approach is analogous to parametric estimation techniques in statistics. In general, these parametric procedures are useful only when the network architecture (i.e. model) is chosen correctly. Too small a network cannot learn the problem well, but a size too large will lead to over-generalization and thus poor performance. Hence, recent studies have sought to optimize network size for a particular class of networks which have the same architecture. There are two general approaches to this optimization problem. One involves using a larger than needed network and training it until an acceptable solution is found. After this, hidden units or weights are removed if they are no longer actively used. Methods using this approach are called *pruning* procedures [37, 47, 55, 68, 76]. The other approach, which corresponds to *constructive* procedures, starts with a small network and then grows additional hidden units and weights until a satisfactory solution is found.

The pruning approach has several shortcomings. Firstly, in practice, one does not know how big a network to start with. Secondly, since the majority of the training time is spent with a network that is larger than necessary, this method is computationally wasteful. Thirdly, many networks with different sizes may be capable of implementing acceptable solutions. Since the pruning approach starts with a large network, it may not be able to find the smallest acceptable solution. Fourthly, these pruning procedures usually measure the change in error when a hidden unit or weight in the network is removed. However, such change can only be approximated[2] for computational efficiency, and hence may introduce large errors, especially when many are to be pruned. Regularization [8, 35, 99] solves some of these problems, but it requires a delicate balance between the error term and the penalty term. It also increases the training time, and the penalty term tends to create additional local minima in which one will frequently get stuck while searching for a "good" solution to the minimization problem [35]. Hence, constructive algorithms seem to be more promising than pruning algorithms.

In this survey paper, we will mainly concentrate on regression problems.[3] In a regression problem, one is given a d-dimensional random vector $\mathbf{X}$, the components of which are called *predictor* variables, and a random variable, Y, called *response*. A regression surface f describes a general relationship between variables $\mathbf{X}$ and Y. The constructive procedure attempts to find a network that is an exact representation or, as is more often the case, a good

[2] Typically, approximation involves using up to the first [47, 68] or second [37, 55] term in the Taylor series expansion for the change in error. Further approximation is possible by computing these values as weighted averages during the course of learning or by assuming that the Hessian matrix of the error function is diagonal [55].

[3] For simplicity, we assume that there is only one target function to be approximated. When there are more than one target functions, each corresponding to one output unit in a neural network, one could approach this by simply treating the approximation of each target function as a different (unrelated) regression problem. Other methods that utilize the relationship among these target functions are discussed in [27].

enough approximation of f. Classification problems can be considered as a special case. For example, for a classification problem with K classes, one can in principle construct a neural network with K output units, whose activation values are usually constrained to [0,1]. The kth output unit ($1 \leq k \leq K$) corresponds to class C_k and learns the posterior probability of class C_k given the input x. A number of other constructive methods that can *only* be applied to classification problems (such as [2, 15, 23, 62]) will not be discussed here. Interested readers may consult the short surveys in [21, 70].

This paper will review both purely constructive procedures and procedures that have constructive components for feedforward neural networks. The rest of this paper is organized as follows. In Section 2, general issues on constructive approximation will be discussed. A taxonomy of constructive procedures based on the resultant network architecture will be presented in Section 3. In Section 4, we will discuss algorithms used to determine the network weights. Criteria to determine when to stop the addition of new hidden units will be summarized briefly in Section 5, followed by some concluding remarks in the last section.

2 Issues in Constructive Approximation

2.1 Defining the Problem as a Function Space Search

The problem of finding a suitable neural network to approximate the target function can be viewed as a function space search problem, with the search space, initial states, goal states, and the control strategy defined by the constructive procedure.

2.1.1 Search Space

The search space is a subspace in a certain function space. The function space is typically a normed linear space[4]. The norm gives us a notion of distance: If w, v are in the normed linear space, then the distance from w to v (or v to w) is $\|v - w\|$. Different norms, and hence different function spaces, are useful in different problems. A few examples will be given in Section 2.1.3.

An exhaustive search in the whole function space is computationally prohibitive. Hence the search space, containing all network functions that can possibly be implemented by the constructive algorithm, is only a subspace and is determined by the way hidden units are generated:

- How the net input to a new hidden unit is computed?

[4] A *normed linear space* [51] is a linear space defined with a real-valued function $\|\cdot\|$ called the *norm*. For any v in the space, the norm satisfies the following three properties:

1. $\|v\| \geq 0$ with equality if and only if $v = 0$.
2. $\|\lambda v\| = |\lambda| \|v\|$ for any scalar λ.
3. $\|v + w\| \leq \|v\| + \|w\|$.

- How a new hidden unit is connected to the existing network?
- What is the transfer function of the new hidden unit?
- How are the new and existing network weights determined?

Note that in contrast to conventional learning with fixed-sized networks, there are two levels of search for constructive procedures. The first level searches for the (nearly) optimal network architecture. With the architecture determined, the second level searches for the (nearly) optimal set of weights.

Usually, the design of the resultant network architecture will have to answer the following questions:

- Whether it is a multi-layer perceptron, a radial basis function network, or others?
- Whether it has a single hidden layer, or multiple hidden layers?
- Whether the connections are regular in the sense that the hidden units are fully connected to all hidden (or input) units in the previous layer and to all hidden (or output) units in the next layer, or there may be irregular connections across the layers?

Most resultant network architectures do have sufficiently strong approximation power (Section 2.2), and hence no representation bias in the machine learning sense. But, on the other hand, constructive methods introduce procedural bias by favoring smaller networks. Moreover, if the target function can be represented sufficiently closely by a reasonably sized network constructed from the procedure, generalization performance can be good. Otherwise, generalization will be poor (Section 2.3). Hence, different constructive procedures, by defining (constraining) different subspaces to be searched, introduce different forms of learning bias [65], and it is important that the particular type of bias matches with the problem at hand. Thus, just as different network architectures favor different types of function [27], there is no constructive procedure that dominates all others over all potential target functions.

2.1.2 Initial State

The initial state refers to the network function before the constructive procedure starts. All procedures start with a network with no hidden units. In certain algorithms [18], the initial network has direct connections from inputs to outputs. Training is first performed on this initial structure, and hidden units are added only when the performance is not satisfactory. This ensures that the regression problem at hand is out of the capability of simple linear regression, or, if it is a classification problem, that the pattern classes are not linearly separable. For the other cases when there is no connection among the (input and output) units, we take this initial state to be the null function.

2.1.3 Goal States

The goal states are network functions that are acceptable solutions to the problem, that is, those that are sufficiently good approximations to the true target function. Whether an approximation is good or not depends on how we measure the "closeness" between functions, which in turn is problem dependent. As mentioned in Section 2.1.1, this distance measure is induced from the norm used in the function space. For two functions f, g defined on an input space X, commonly used distance measures include the uniform distance,

$$\|f - g\| = \sup_{x \in X} |f(x) - g(x)|. \tag{1}$$

If f is the target function to be learned and g is the actual function implemented, the uniform distance singles out x values at which the approximation is worst (that is, where the absolute error is greatest) and assigns as a measure of closeness these worst possibilities. It thus provides absolutely certain bounds on the error at the expense of these bounds having to be large enough to be valid at every point, no matter how exceptional.

Another commonly used measure is the L_p distance:

$$\|f - g\|_p = \left[\int_X |f(x) - g(x)|^p d\mu(x)\right]^{1/p}, \tag{2}$$

where μ is the input environment measure, and p is a real number, $p \geq 1$. In the case $p = 2$, we recover the usual Euclidean distance. Note that as in both (1) and (2), the distance measure computed has to be based on the whole input space X, not just on the patterns in the training set. The implications will be re-examined in Section 2.2.

Moreover, there are more than one goal states, i.e., there exist many networks that are *sufficiently* good approximators. The good side is that it is usually not necessary to find the globally best network, and most constructive procedures just give one of these sufficiently good approximators as solution. The bad side is that different network solutions, though equally good as approximators, may have different interpretations. It is sometimes desirable to examine multiple solutions before selecting the one that explains the data best. This may be done, for example, by repeating the constructive procedure several times, using different initial weights in the learning process.

2.1.4 Control Strategy

The generate-and-test methodology is taken in all the constructive procedures:

1. Generate a possible network according to the control strategy, which defines the way to search for a solution (architecture and weights).

2. Test to see if this solution is acceptable.

3. If it is acceptable, exit; otherwise, go back to step 1.

The search for the best set of weights can be performed using various optimization algorithms, which will be discussed in Section 4.2. For the architecture, constructive procedures always search the smaller networks first, and then extend to larger and larger networks. Besides having better generalization performance, smaller networks also enjoy the following advantages:

- The cost of computation, measured by the number of arithmetic operations, is of $O(W)$ on a serial computer, where W is the number of weights. Hence, a smaller network is more efficient both in forward computation and in learning.
- A smaller network can be more easily described by a simpler set of rules. Moreover, the function of each hidden unit may be more easily visualized.

With a given number of hidden units, certain constructive procedures can only produce one network architecture [3, 18, 39, 43, 57, 75, 88, 91, 104], and hence there is only a single network architecture that can possibly be generated given the current network configuration, and so the control strategy is simple. In more powerful procedures, however, there are several such possibilities and some kind of searching technique is required. This allows for greater exploration of the function space, possibly leading to better architectures found, at the expense of longer time. The following search strategies have been employed.

2.1.4.1 Depth-First Search with Backtracking

Depth-first search, one of the most straightforward ways to implement systematic generate-and-test procedures, is used in [69]. However, depth-first search is an exhaustive search technique. For it to be effective, one should ensure that unproductive partial paths are never too long. This is done in [69] by defining certain networks as not "promising", backtracking is then used to recover from these dead-end positions.

2.1.4.2 Best-First Search

Teng and Wah [96] proposed a population-based learning system for designing feedforward networks. Under this scheme, a pool of networks, each having no hidden unit initially, is maintained. The most promising network is selected, trained and allowed to grow for a certain time quantum. Its performance, based on a number of factors such as the training error, its rate of change and also the number of epochs trained, is then measured. This cycle repeats again until a satisfactory network solution is found. Note that as learning progresses, the pool may have networks with different numbers of hidden units.

2.1.4.3 Nondeterministic Search

For example, in [97], a new architecture is randomly chosen from the set of possible architectures. An energy criterion, modified from the *minimal description length* (MDL) [79] information criterion, is then computed for the new network and the question of whether to accept or reject this network is determined via simulated annealing.

2.1.4.4 Genetic Algorithms

A related approach is the use of evolutionary approaches. By viewing the search for the optimal architecture as searching a surface defined by levels of trained network performance above the space of possible network architectures, [64] mentioned that such a surface is typically

- infinitely large, as the number of possible hidden units and connections is unbounded;
- non-differentiable, since changes in the number of units or connections must be discrete, and can have a discontinuous effect on network performance;
- complex and noisy, as it is dependent on initial conditions;
- deceptive, as structurally similar networks can have very different network performance, and
- multi-modal, as structurally dissimilar networks can have very similar performance.

The main idea of using genetic algorithm based evolutionary approaches is then that they are good at dealing with these kinds of search spaces. A review of evolutionary artificial neural networks (EANN) can be found in [100, 103]. However, note that in the direct encoding scheme for EANN connectivity (such as in [64]), the size of the connectivity matrix has to be pre-defined, and hence they cannot be used with constructive procedures. Moreover, these methods, besides being very demanding in time and storage, also require a good representation of the network structure and a good design of the genetic operators.

2.2 Universal Approximation and Convergence Properties

Given a constructive algorithm, we need to consider the universal approximation capability of the resulting network architecture: Is the family of functions implemented by the network broad enough to contain f or a good enough approximation of f? The next question concerns the convergence properties: Can the procedure produce a convergent sequence of network functions that, in the limit, approximates the target function as closely as desired. In other words, does the sequence $\{f_n\}$ so generated strongly converges[5] to f as $n \to \infty$? Apparently, the universal approximation capability of a network structure is a prerequisite for the convergence of its learning procedure. Attempts to solving the problem without considering these questions could be very time-consuming if not fruitless.

Note that, as mentioned earlier, the norm used in the convergence definition has to be based on the whole input space X, not just on the training patterns as is done in [11, 60]. This is because one is usually more interested in the generalization performance rather than performance on the training set, and a perfect recall of the training set is always possible simply by having more hidden units. Hence, "convergence", in the sense of reducing the training error to zero [11, 60], is not quite interesting.

[5] A sequence $\{f_n\}$ *strongly converges* [51] to f if $\lim_{n\to\infty} \|f - f_n\| = 0$, where $\|\cdot\|$ is the norm for the function space being considered, such as those defined in (1) and (2).

2.3 Bias-Variance Tradeoff

Denote the function estimator corresponding to a neural network with n hidden units as f_n. The *mean squared error* (MSE) [32] of f_n at an input point x can be decomposed into two components:

$$\begin{aligned} MSE\{f_n(x)\} &= E[(f(x) - f_n(x))^2] \\ &= (f(x) - E[f_n(x)])^2 + E[(f_n(x) - E[f_n(x)])^2] \\ &= [\text{bias}(x)]^2 + \text{variance}(x), \end{aligned}$$

where $E[\cdot]$ is the mathematical expectation taken over all training samples of size N. The *integrated mean squared error* (IMSE) is obtained by integrating MSE(x) over all x's in the input domain X. As empirically shown in [32], bias falls and variance increases with the number of hidden units. This phenomenon agrees well with what one generally observes in practice. When a neural network is trained, the error on the training patterns continues to decrease. The IMSE, on the other hand, tends to decrease in step with the training error initially, as the network generalizes from the training data to the underlying function. At some point, however, the network starts to take advantage of these idiosyncrasies in the training data and the IMSE starts to increase again even though the training error continues to decrease (Figure 1). Hence, it is important to have a proper tradeoff of the bias and variance by stopping the network growth appropriately [32].

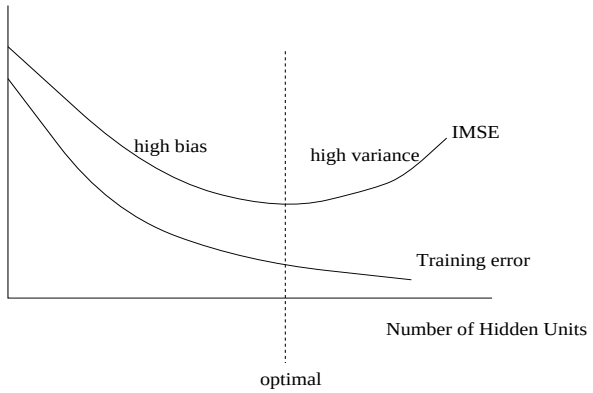

Figure 1: Typical error curves with increase in the number of hidden units.

3 Network Architecture

This section describes the various constructive procedures, categorized by the resulting network architectures they produce. The approximation capabilities of these architectures are also discussed. One can see that an important goal of these constructive procedures is to develop powerful feature detectors without using excessive weights or hidden units. Pruning

is sometimes used in the procedures mentioned in this section. The network may be pruned after completion of the constructive process [39, 43], or be interleaved with the constructive process [69]. However, details of these pruning methods will not be discussed in this paper.

3.1 Single Hidden Layer

New hidden units are always fully connected to all input and output units, with the resultant network having one single hidden layer (Figure 2). Because of the regular network structure and constant fan-in for the hidden units, hardware implementation is simple. The hidden units may be of simple or complex functional forms.

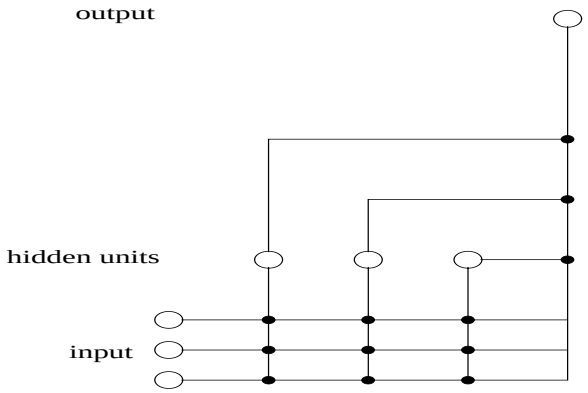

Figure 2: Single hidden layer architecture. In the diagram, empty circles represent input/hidden/output units while a black dot refers to a connection between units.

3.1.1 Simple Hidden Units

There are two main categories. The first category is based on the *multi-layer perceptron* (MLP), like the *dynamic node creation network* [3] and [39, 49, 66, 86, 91, 104, 106]. The hidden unit transfer function in a MLP is of the form:

$$g(\mathbf{x}) = \phi(\mathbf{a}^T\mathbf{x} + \theta),$$

where ϕ is usually the sigmoidal function $\phi(z) = 1/(1 + e^{-z})$.

The other category is based on the *radial basis function network* (RBFN), like the *Gaussian potential function network* (GPFN) [57], the *resource-allocating network* (RAN) [75], and the supervised *growing cell structures* (GCS) [29]. The hidden units, also called *radial basis functions* (RBF), are of the form:

$$g(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{x}_i\|),$$

where $\|\cdot\|$ is usually the Euclidean norm or a weighted norm [33] and $\mathbf{x}_i$ is the *center* of the basis function. ϕ is a radial kernel function, the most common form of which is the Gaussian

function:

$$\phi(z) = \exp(-\frac{z^2}{2\sigma^2}),$$

where σ is called its *width*.

Coarse-coding resource-allocating network [14] is similar to RAN, but replaces the layer of multi-dimensional Gaussian basis functions by a layer of one-dimensional Gaussian basis functions and a layer of pi-neurons. The multiplication operation of the pi-neurons basically mimics the multi-dimensional units using its one-dimensional counterparts, but sharing of the one-dimensional units among different pi-neurons is allowed, resulting in more compact networks and hence more efficient learning.

The approximation properties of these two types of network are well known. Universal approximation of single hidden layer MLP using different distance measures has been extensively discussed in the literature (see for example [13, 31, 40, 41]), while that of RBFN can be found in [36, 71, 72]. Intuitively, the results mean that these two types of network, with as few as one hidden layer, can approximate any "reasonable" real-valued function to any desired accuracy, provided that sufficiently many hidden units are available.

In practice, the two networks may have different performances in different problems. While a RBF unit responds significantly to only a small localized region of the input space, a hidden unit in a MLP has spreads that are infinitely large along directions orthogonal to the vector $\mathbf{a}$. Hence, RBFN may not perform well for high-dimensional data, because of the "curse of dimensionality" [6] which arises from the fact that data in high-dimensional space are surprisingly sparse [28]. Single hidden layer MLP, on the other hand, may not work well for certain kinds of highly nonlinear problems, like the two-spirals problem [18].

3.1.2 Flexible Hidden Units

In this category, the network constructed still has only a single hidden layer, but the hidden unit transfer functions are more "flexible" and are capable of forming more complicated feature detectors [67]. This flexibility is achieved at the expense of requiring additional parameters in each hidden unit transfer function. Thus, for the same number of hidden units, this kind of networks will have a smaller bias but a larger variance than those with simple hidden units.

3.1.2.1 Adaptive Spline Networks

The first method is the *adaptive spline networks* (ASP) [25], which is developed from the statistical technique *multivariate adaptive regression splines* (MARS) [26]. The regression surface is represented in the form:

$$f_n(x) = \sum_{j=1}^{n} \beta_j g_j(x),$$

where β_j is the weight connecting the output to the jth hidden unit g_j, which is of the form:

$$g_j(x) = \prod_{k=1}^{K_j} (x_{j(k)} - t_{kj})_+^q.$$

Here $1 \leq j(k) \leq d$ selects an input component, $1 \leq K_j \leq d$ is the number of factors in the product, t_{kj} is the knot location, and q is the order of the spline. $(x_{j(k)} - t_{kj})_+^q$, known as the truncated power function, is defined as:

$$(x_{j(k)} - t_{kj})_+^q = \begin{cases} 0 & x_{j(k)} \leq t_{kj}, \\ (x_{j(k)} - t_{kj})^q & x_{j(k)} > t_{kj}. \end{cases}$$

MARS, and hence also ASP, can realize both additive functions and binary tree topology similar to those generated by recursive partitioning strategies like the *classification and regression tree* (CART) [7]. Both additive and CART methods have been highly successful in many complementary situations: additive modeling when the target function is close to additive, and CART when it has significant high order interactions between the input variables. Thus, MARS and ASP are likely to be successful in a wide range of situations.

3.1.2.2 Projection Pursuit

The second class of methods is also related to a statistical technique, called *projection pursuit regression*[6] (PPR) [28, 42]. In PPR, the regression surface is approximated by a sum of n empirically determined univariate functions g_j of linear combinations of the predictors, i.e.

$$f_n(x) = \sum_{j=1}^{n} g_j(\mathbf{a}_j^T \mathbf{x}), \tag{3}$$

where $\mathbf{a}_j$ is the projection vector with $\|\mathbf{a}_j\| = 1$. The g_j's are called *smoothers* in the statistics literature. The obvious similarity between (3) and single hidden layer neural networks, by taking the g_j's as hidden unit transfer functions, suggests that a direct formulation of PPR into neural network is possible, such as in projection pursuit learning (PPL) [43, 52].

This class of methods can be further sub-categorized depending on whether the hidden unit transfer function is nonparametric or parametric.

3.1.2.2.1 Flexible Nonparametric Hidden Units

In this category, the hidden unit transfer functions are nonparametric, i.e., they are not restricted to any pre-specified family of functions. In PPR, g_j is obtained by a variable span smoother called the supersmoother [95], whose basic building block is a symmetric k-nearest neighbor linear least squares fitting procedure. The optimal value of k, called the *span* of the supersmoother, is chosen for each input point using a local cross-validation procedure. However, the use of supersmoothers suffers from several disadvantages, as discussed in [43]:

[6] PPR has also been applied to classification problems, see [22, 81].

1. the use of large regression tables,

2. unstable approximation in calculating derivatives, and

3. piecewise interpolation in computing activation values.

Automatic smoothing spline projection [82] uses smoothing splines [95], which are also non-parametric, as smoothers. They can give accurate derivative calculation and smooth interpolation, but still require the use of a smoother matrix. Moreover, the generalized cross validation statistic, which is used to select the degree of smoothness, tends to under-smooth [38]. Consequently, heuristics are required to remedy this problem. On the computational aspect, smoothing splines are usually more computationally intensive.

The universal approximation capability of PPR networks, with respect to the L_2 norm, follows directly from its convergence property, which will be discussed in Section 4.1.3.

3.1.2.2.2 Flexible Parametric Hidden Units

Here, the hidden unit transfer function is of a certain parametric form, instead of non-parametric, though the form is still very flexible. Compared to the use of nonparametric hidden units in PPR, this method enables smooth interpolation, instead of piecewise interpolation. Moreover, the derivative of the hidden unit transfer function is usually simple, enabling fast and accurate computation of the derivatives without the use of large regression tables. However, the flexibility or complexity of the hidden units is controlled by a parameter. The selection of this parameter may be very critical and a wrong selection may lead to poor training and generalization performance.

In *projection pursuit learning* (PPL) [43], the hidden unit transfer function is represented as a linear combination of the Hermite functions:

$$g(z) = \sum_{r=1}^{R} c_r h_r(z),$$

where $h_r(z)$'s are the orthonormal Hermite functions and R, called the *order*, controls the complexity of the hidden unit. This parameter is less crucial to performance when a bias term is added to the linear combination of the predictor variables [52].

Another method is to use fixed-sized neural networks as hidden units. For example, *connectionist projection pursuit regression* [98] uses sigmoidal networks. Saha *et al.* [83] use radial basis function networks. The number of hidden units in these fixed-size networks controls their flexibility. The universal approximation capabilities for these networks and PPL networks follow directly from the universal approximation results for traditional feedforward networks, provided that a bias term is included in the linear combination of the predictor variables [52].

In *ridge polynomial networks* (RPN) [88], each hidden unit is a *pi-sigma network* (PSN) [87]:

$$g(\mathbf{x}) = \prod_{i=1}^{k} (\mathbf{a}_i^T \mathbf{x} + \theta_i),$$

where k is called its *degree*. Hidden units added to the network have increasing degrees, i.e., the first hidden unit is a PSN of degree one, the second one is of degree two, and so on. Thus, subsequent hidden units have an increasing number of parameters associated and become increasingly flexible. The RPN is a sum of these PSNs,

$$f(x) = \sum_{j=1}^{n} g_j(\mathbf{x}) = \sum_{j=1}^{n} \prod_{i=1}^{j} (\mathbf{a}_{ji}^T \mathbf{x} + \theta_{ji}).$$

Its universal approximation capability, with respect to the uniform norm, is mentioned in [88].

3.2 Multiple Hidden Layers

In this category, the resultant network has more than one hidden layers. Moreover, there may be irregular connections across different layers. This flexibility over single hidden layer networks allows powerful high-order feature detectors to be developed.

3.2.1 Full Connection to All Pre-existing Units

In the *cascade-correlation architecture* [18], the hidden unit transfer function is still of simple form, such as sigmoidal or radial basis functions. However, when a new hidden unit is to be created, besides establishing connection with each of the network's original inputs and outputs, it also establishes connection with every pre-existing hidden unit. Each new unit therefore adds a new one-unit "layer" to the network, lending to a cascade architecture (Figure 3). Cascade-correlation architecture has been applied to segmentation of magnetic resonance images of the brain [34], prediction of software reliability [48], classification of cervical cells [63], implementation of the inverse kinematic transformations of a robot arm controller [90], modeling of human cognitive development on balance scale problems [89], and modeling of chaotic timeseries [12]. A parallel implementation of the original cascade-correlation architecture using the Time Warp Operating System is also available [93].

The resultant deep network structure leads to the creation of very powerful high-order feature detectors even with simple hidden unit transfer function. This is sometimes very advantageous. However, it also gives rise to long propagation delays and an ever-increasing fan-in of the hidden units as more units are added. All these make VLSI implementation difficult. Generalization could also have problems.

There are many variants of this basic architecture. For example, in [90], instead of having the hidden units connected to all output units, each output unit has its own set of hidden units. It is reported to have faster training and better generalization, at the expense of using more hidden units. In the *cascade network architecture* [59], after a hidden unit g is trained, nonlinear functions of g are also added to the network together with g. The cascade structure may also be combined with hidden units of more powerful transfer function such as those mentioned in Section 3.1.2. For example, the hidden units in the *cascade LLM networks* [58] are local linear maps. Each such hidden unit learns a linear approximation of the target function that is valid within its "receptive field".

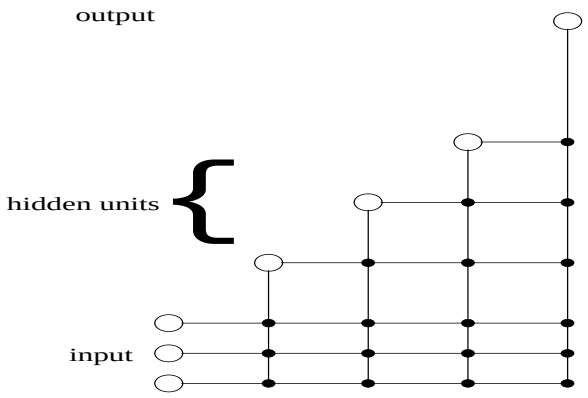

Figure 3: Multiple hidden layer network with full connection to all pre-existing units.

The universal approximation capability is obvious, as this cascade architecture can be reduced to the regular single hidden layer network by removing all the hidden-to-hidden weights.

3.2.2 Partial Connection to Some Pre-existing Units

To avoid the problems resulted from full connection, one may use pruning to reduce the degree of connectivity. In [92], as each hidden unit is trained, the saliency of its weights is calculated and the weights that are determined to be weak are eliminated. However, such an approach is computationally expensive and the saliency can only be approximated, sometimes with high error. A natural alternative which will be discussed in this section is to limit the fan-in of the hidden units and connect the new hidden units to only a selected few of the pre-existing units.

3.2.2.1 Fixed Connection Policy

The very deep structure in the cascade-correlation architecture is caused by many one-unit hidden layers. Phatak and Koren [74] modified it by allowing more than one hidden units in each layer, and instead of connecting the new hidden units to all pre-existing hidden units, they are connected only to the previous hidden layer. The fan-in of the hidden units can thus be controlled by restricting the number of units allowed in each hidden layer. When this maximum is reached and a new hidden unit is to be added, the output layer is first collapsed into a new hidden layer, with the new hidden units added to this new layer. As a result, a regular multiple hidden layer structure is formed (Figure 4).

However, while the hidden units in the higher layers (i.e. those that are closer to the output layer) are capable of implementing complex functions in the input space, simpler functions may be difficult to realize when no direct input to hidden connections are provided in the architecture. Moreover, the maximum number of hidden units allowed in each layer may be crucial. Restricting this to a small number limits the ability of the hidden units to

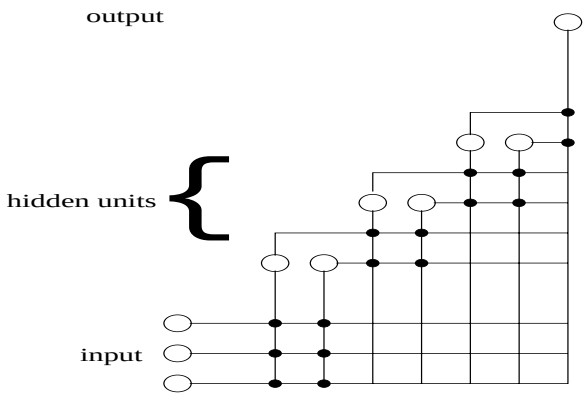

Figure 4: Modified cascade-correlation architecture with partial connection.

form complicated feature detectors, as each hidden unit can only see its previous hidden layer outputs. With this restriction in mind, the universal approximation capability of this kind of architecture, even when the number of hidden layers is allowed to grow without bound, seems unclear. This weakness, however, can be removed by re-introducing connections from all hidden units to the input and output units.

A similar method is the *stacking learning algorithm* [19], which also produce hidden layers by collapsing the output layer. The hidden units have direct connections to all input and output units, hence the resultant network architecture is an universal approximator. However, additional hidden units are not added to these collapsed layers, i.e., the number of hidden units in each hidden layer is always equal to the number of output units. This restriction possibly impedes the network performance.

3.2.2.2 Competitive Connection Policy

Another class of algorithms which also produces multiple hidden layers is inspired from the *group method of data handling* (GMDH) [20]. Here, instead of pre-determining the way in which new hidden units are connected, a lot of candidate units using different connections are generated and the new hidden unit is then chosen from among these candidate units in a competitive manner.

In [73], new connections may be created by any possible combination of the outputs from the preceding hidden (or input) layer. Candidate hidden units are then added to the network whenever their performances satisfy certain performance measure. In the *self-organizing neural network* [97], candidate hidden units are generated by allowing connections to any input or pre-existing hidden unit. A new hidden unit is then randomly selected from this candidate pool and simulated annealing is used to determine whether to accept or reject this addition. The resulting architecture is no longer a regular multiple hidden layer structure, but has many intra-layer connections (Figure 5).

Because of the large number of candidate units that are to be created, the hidden unit transfer functions are usually constrained to be of some simple form for computational reasons

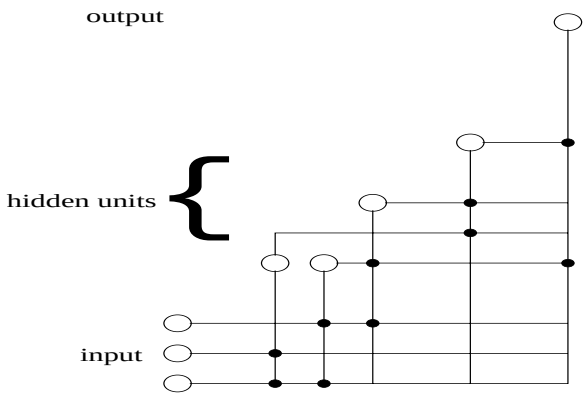

Figure 5: GMDH-type architecture.

(details in Section 4.1.5), of which the polynomial form is the commonest, such as:

$$g(i,j) = a + b_i o_i + b_j o_j + c_i o_i^2 + c_j o_j^2 + d_{ij} o_i o_j,$$

where the a, b, c, d's are the connection weights, and i, j represent the indices of the units where connections are made, and o_i, o_j are the corresponding output activations from these selected units.

3.2.3 Other Ad hoc Methods

For example, in [69], a whole hidden layer may be added to the network between the last hidden layer and the output layer. The number of hidden units in the new layer n_{new} is heuristically set to be:

$$n_{new} = \lfloor \frac{n_o + n_h}{2} \rfloor,$$

where n_o is the number of output units and n_h is the number of hidden units in the last hidden layer. Obviously, this method is quite *ad hoc*.

4 Learning Algorithm

4.1 Training of the Network Weights

To add a new hidden unit, the weights of both the new and existing connections have to be determined. This section summarizes the methods that are commonly used, and also their convergence properties. Note that computational concern is usually an important issue. If the network has a fixed size, its weights have to be optimized only once. However, in constructive algorithms, this optimization process has to be repeated each time a hidden unit is added. In some cases, each new hidden unit has a number of candidates and optimization is required for each candidate. Hence, to be computationally efficient is very important.

4.1.1 Encoding the Novel Pattern

For algorithms that can be used for sequential learning, the network weights may simply be obtained from the training pattern. For example, in RAN[7] [14, 75], which uses radial basis functions as hidden units, the network allocates a new hidden unit to memorize a novel pattern. A pattern is considered novel when it is far away from all basis units in the network and its error, which is the difference between the desired output of the pattern and the network output, is large. Another novelty detection method is discussed in [80]. The center of the new unit is set to the novel pattern, with its width proportional to the distance from the pattern to the center of the nearest basis function. The weights connecting this new unit to the output is set to the error vector. On the other hand, when the pattern is not novel enough, no new unit is added and the parameters of the nearest basis function and its weights to the output are updated accordingly to reduce the error. GPFN [57] also uses a similar method. The convergence properties of these procedures are not known.

As mentioned in [29], the problem with this approach is that noisy data may lead to a large number of hidden units. Moreover, there is a resolution parameter that specifies the smallest distance two RBF units can have and features that are smaller than that will get averaged out. This makes it impossible to model functions appropriately that vary on a smaller scale.

4.1.2 Retraining the Whole Network

The next simplest approach is to use nonlinear optimization techniques such as the well-known backpropagation algorithm to train the whole network after hidden unit addition [3, 39]. Convergence of the network functions to the true function using this method follows directly if the universal approximation capability of the network architecture holds.

However, optimization becomes increasingly difficult and slow as the network grows in size. Moreover, learning all the weights at the same time suffers from the so-called *moving target* problem [18]. Intuitively, each hidden unit is trying to evolve into a useful feature detector, but its task is greatly complicated by the fact that all the other hidden units are changing at the same time. The hidden units in a given layer of the network cannot communicate with one another directly; each unit sees only its inputs and the error signal propagated back to it from the network's outputs. The error signal, defining the problem that the unit is trying to solve, changes constantly. Instead of a situation in which each unit moves quickly and directly to assume some useful role, we see a complex dance among all the units that takes a long time to settle down.

4.1.3 One Unit at a Time

A less computationally intensive method that also avoids the moving target problem is by keeping the weights of the pre-existing hidden units fixed (*input weight freezing*), while allowing only the new hidden unit and the weights connecting hidden and output units to learn. This is commonly used in many algorithms, such as PPL-type algorithms [28, 43, 82, 83, 98],

[7] A function space approach to analyzing the learning algorithm of RAN is developed in [46].

cascade-correlation architecture and its variants [18, 59, 58, 91, 104, 105], and methods in [86, 106]. Experimentally, this strategy allows the network to learn faster. Moreover, the hidden units, acting as feature detectors, are never cannibalized once built. They are available from that time on for producing outputs or more complex features.

However, because the parameters of the hidden units are kept fixed after installation into the network, the convergence property of the constructive algorithm does not follow readily from the universal approximation capability of the network architecture. In some cases, convergence can still be shown to exist. For example, strong convergence of PPR is proved in [45], which states that if each new g_n in (3) at stage n is given by the conditional expectation [42]:

$$g_n(z) = E(f - f_{n-1} | \mathbf{a}_n^T \mathbf{X} = z),$$

and the projection direction $\mathbf{a}_n$ is chosen as long as

$$E(g_n(\mathbf{a}_n^T \mathbf{X}))^2 > \rho \sup_{\mathbf{b}^T \mathbf{b}=1} E(g_n(\mathbf{b}^T \mathbf{X}))^2,$$

where $0 < \rho < 1$ is fixed, then f_n in (3) strongly converges to the desired f. Convergence properties of the other algorithms will be discussed in later sections.

To make up for the possible degradation in performance, back-fitting [28, 43] is sometimes used to fine tune the existing hidden units when a new hidden unit has been added. This consists of cyclically adjusting the parameters associated with each previously installed hidden unit, while keeping the parameters of the other units fixed, until there is no significant change. Alternatively, training of the new hidden unit(s) can be followed by retraining the whole network, as in the *sequential network construction* method [66].

Training of the new hidden unit can be performed by back-propagation as usual. However, for efficient modeling of discontinuities in the target function, [56] used data from different localized portions of the input space to train each new unit. It is reported to have more accurate results and require considerably less computation when compared with standard back-propagation, for functions with many discontinuities.

4.1.4 One Unit and One Layer at a Time

Training can often be speeded up by proceeding in a layer-by-layer manner. First, the weights feeding into the new hidden unit are trained (input training). They are then kept constant and the weights connecting from it to the output units are trained (output training). In so doing, only one layer of weights needs to be optimized each time and there is never any need to back-propagate the error signals.

There are several criteria which can be used in input training. For example, in [59], the new hidden unit is treated as an interim output unit, and the usual (squared) error criterion is used. In the cascade-correlation architecture [18] and its variant [91, 104], the new hidden unit maximizes a correlation function between the residual error and the hidden unit activation. Some other correlation-based functions are proposed in [11, 30, 53]. The convergence property of the cascade-correlation learning procedure, for networks using the

hyperbolic tangent as hidden unit transfer function and assuming the input environment measure to be uniform, is proved in [16]. More general results, extending to other hidden unit transfer functions, correlation functions in [30, 53] and input environment measures, are discussed in [53]. However, for the criterion function in [11], its convergence property is not known.

Besides, a projection index [105], which finds "interesting" projections that deviate from Gaussian distributions, can be used. However, this criterion is probably more suitable for exploratory data analysis [24] rather than for regression, as projections that are of the Gaussian form may also be useful. Its convergence properties is, again, unknown.

Alternatively, one can use the same error criterion in both input and output training (such as the squared error criterion), such as in [59, 58]. One can also use a scheme similar to that proposed in [1]. The weights in each layer are updated in turn, by keeping all the other weights unchanged, and the whole process is cycled many times. For example, in PPL [43], the order of updating is first the projection directions, then the parameters associated with the hidden unit transfer function, next follows the output weights, then back to the projection directions, and so on. Note that only the optimization of the projection directions is a nonlinear problem, while the other two are linear.

In RPN, there is only one single layer of weights for each PSN, so all the weights can be updated at the same time. However, this updating scheme can lead to instability problems unless the learning rate is sufficiently small [88]. To remedy the problem, an asynchronous rule is used. If the new hidden unit (which is a PSN) is of degree k, then one of the k projection vectors is chosen and updated according to the squared error criterion. For the same input pattern, the output is recomputed with the updated projection vector, and the error is used to update another projection vector. This procedure is performed k times for every input so that all k sets of weights are updated once. The convergence of this procedure is unknown.

4.1.5 Reducing to a Linear Problem

An entirely different method is to use a restricted class of transfer functions so that the hidden unit output is linear with respect to its parameters, as is common in GMDH-type networks [73, 97]. Optimizing the parameters with respect to the squared error criterion reduces to a linear problem, i.e. computation of the pseudo-inverse with closed-form solution. The speed thus gained allows repetition of this optimization problem for each connectivity pattern during unit addition.

4.1.6 Miscellaneous Scheme

In supervised GCS [29], adaptation of the parameters associated with the RBF units is similar to that of Kohonen's feature map [50], while that of the weights connecting the RBF units to the output units is by the delta rule. When a new unit is to be added (which is determined by resource variables distributed among the RBF units), the new weights are obtained from units in its neighborhood.

4.2 Optimization Algorithm

Training can be done by any nonlinear optimization algorithm that works for fixed-size networks, such as back-propagation or its variants like quickprop [17]. Alternatively, standard optimization methods such as quasi-Newton, conjugate gradient, Newton-Raphson, Gauss-Newton or Levenberg-Marquardt methods [84] or other methods like recursive least-squares [4], extended Kalman filter [46] and genetic algorithms [101] can also be used. A review can be found in [44, 102].

In algorithms that have the optimization problem boiled down to a linear problem, the optimal weights may simply be computed by the pseudo-inverse, instead of the methods mentioned above.

4.3 Use of Patience Parameter

For those constructive algorithms in which the optimization problem boils down to a linear problem during training, the connection weights can be obtained fairly easily. Otherwise, it is computationally more effective to stop the training of each new hidden unit prematurely, through the use of a *patience* parameter as is done in [3, 18, 39, 104]. Training is stopped when the training error does not decrease by a certain fraction after a pre-specified number (the patience) of training epochs. Experimentally, the use of patience can save a substantial amount of training time, compared to running to a fixed number of training epochs determined *a priori*, without degrading the overall performance [94].

4.4 Use of Candidates

The use of candidates during training of the new hidden unit is common in many constructive algorithms. A number of candidates are generated, from which the "best" one is selected to be installed into the network. Training of the different candidates may start from different initial seeds, hence a large candidate pool gives a better chance of finding a good local optimum in the nonlinear optimization problem. In a way, this corresponds to the Monte Carlo sampling of the weight space to get the best solution to the problem. For GMDH-type algorithms, candidates are a result of the competitive connection policy, in which different candidates using different combinations of connections or different transfer functions are generated. Candidates are also used to allow for the selection for different hidden unit transfer functions in the cascade-correlation architecture [18], and for different orders of the hidden unit transfer function in PPL [54].

On the selection of the candidate, the winner may simply be the one producing the smallest residual error. However, more complex hidden units may be able to produce smaller training error, but with worse generalization performance. Hence, information criterion such as MDL [79] may also be used in the selection process.

5 Criterion to Stop Adding New Hidden Units

This involves the issues of evaluating network performance, and finding the architecture with the best expected performance. Model selection is a big topic, and we give only a brief summary here. For example, it may be based on the training error, when it is less than a threshold [39] or flats out. However, it is well known that the training error is biased. Alternatively, it may be based on a separate test set [73], or on more complicated cross-validation [66] or bootstrapping methods. Criteria like *Akaike's information criterion* (AIC), *Bayesian information criterion* (BIC), *final prediction error* (FPE), *generalized cross-validation* (GCV), *generalized prediction error* (GPE), *predicted squared error* (PSE) etc. [66] may also be used. Besides, the Bayesian evidence [61], by formulating in a Bayesian framework, may also be used.

6 Conclusion

In this paper, we review the different procedures used for constructing feedforward neural networks. Although the list of articles surveyed here is not exhaustive, one can still notice a conglomeration of various network architectures and learning algorithms. All constructive procedures attempt to construct good feature detectors, i.e. the hidden units, for the specific problem under investigation, without using excessive weights or hidden units. While procedures that construct networks with a single layer of simple hidden units may not be able to solve certain highly nonlinear problems, other procedures develop powerful feature detectors either by increasing the complexity of the hidden unit transfer function or by building a multi-layer structure. In the process, different procedures induce different learning biases, and hence different constructive procedures are suitable for different types of target function to be learned. The identification of what class of target functions is most suitable for a particular constructive procedure will be particularly useful from a practical point of view.

Moreover, computational efficiency is always an important concern. One may also notice a close relationship between the hidden unit transfer function, the number of possibilities to connect the new hidden unit, and the learning algorithm. Training of complex hidden units is usually more time-consuming than the simple ones, and hence the learning algorithm for network weights sometimes requires freezing the previously installed hidden units. Restricting the hidden unit transfer function to parametric form, and learning in a layer-by-layer manner, are also tricks to make the computation more efficient. In the case when many different connection patterns are tried for a new hidden unit, as in GMDH-type algorithms, the hidden unit transfer function may have to be further simplified to be of the polynomial form.

Besides, the close link between constructive procedures and forward stepwise regression techniques in statistics is worth mentioning. As discussed before, a number of the constructive procedures have been inspired by statistical techniques like MARS, PPR, and GMDH. The close relationship between various statistical methodologies (like discriminant analysis, regression and cluster analysis) and neural network models has been discussed in [5, 9, 10, 27, 77, 78, 85]. A number of other nonlinear regression techniques may possibly also have neural network formulations.

Also, although the approximation capabilities of many network architectures have been examined extensively in recent years, the convergence properties of most constructive procedures are still unknown. This definitely deserves more attention in the future.

7 Acknowledgments

This research has been partially supported by the Hong Kong Research Grants Council under grant RGC/HKUST 15/91 and the Hong Kong Telecom Institute of Information Technology under grant HKTIIT 92/93.002. The first author is also supported by the Sir Edward Youde Memorial Fellowship.

References

[1] S. Abe. Learning by parallel forward propagation. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 99–104, San Diego, June 1990.

[2] E. Alpaydin. GAL: networks that grow when they learn and shrink when they forget. TR 91-032, International Computer Science Institute, May 1991.

[3] T. Ash. Dynamic node creation in backpropagation networks. *Connection Science*, 1(4):365–375, 1989.

[4] M.R. Azimi-Sadjadi, S. Sheedvash, and F.O. Trujillo. Recursive dynamic node creation in multilayer neural networks. *IEEE Transactions on Neural Networks*, 4(2):242–256, March 1993.

[5] A.R. Barron and R.L. Barron. Statistical learning networks: A unifying view. In E. Wegman, editor, *Computing Science and Statistics, Proceedings of the 20th Symposium Interface*, pages 192–203, Washington, DC, 1988. American Statistical Association.

[6] R.E. Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, NJ, 1961.

[7] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth, Pacific Grove, California, 1984.

[8] Y. Chauvin. A back-propagation algorithm with optimal use of hidden units. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 519–526. Morgan Kaufmann, San Mateo, CA, 1989.

[9] B. Cheng and D.M. Titterington. Neural networks: A review from a statistical perspective (with discussion). *Statistical Sciences*, 9(1):2–54, 1994.

[10] V. Cherkassky. Neural networks and nonparametric regression. In *Proceedings of the IEEE-SP Workshop*, pages 511–521, Helsingoer, Denmark, August 1992.

[11] P. Courrieu. A convergent generator of neural networks. *Neural Networks*, 6(6):835–844, 1993.

[12] R.S. Crowder. Predicting the Mackey-Glass timeseries with cascade-correlation learning. In *Proceedings of the Connectionist Models Summer School*, pages 117–123, 1990.

[13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signials and Systems*, 2:303–314, 1989.

[14] G. Deco and J. Ebmeyer. Coarse coding resource-allocating network. *Neural Computation*, 5(1):105–14, 1993.

[15] G. Deffuant. Neural units recruitment algorithm for generation of decision trees. In *Proceedings of the 1990 IEEE International Joint Conference on Neural Networks*, volume 1, pages 637–642, San Diego, CA, USA, June 1990.

[16] G.P. Drago and S. Ridella. Convergence properties of cascade correlation in function approximation. *Neural Computing & Applications*, 2:142–147, 1994.

[17] S.E. Fahlman. Faster learning variations on back-propagation: An empirical study. In D.S. Touretzky, G.E. Hinton, and T.J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, Los Altos, CA, 1988. Morgan Kaufmann.

[18] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, Los Altos CA, 1990.

[19] W. Fang and R.C. Lacher. Network complexity and learning efficiency of constructive learning algorithms. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 366–369, Orlando, Florida, USA, June 1994.

[20] S.J. Farlow, editor. *Self-Organizing Methods in Modeling : GMDH Type Algorithms*, volume 54 of *Statistics: Textbooks and Monographs*. Marcel Dekker, Inc., New York, 1984.

[21] E. Fiesler. Comparative bibliography of ontogenic neural networks. In *Proceedings of the International Conference on Artificial Neural Networks*, volume 1, pages 793–796, Sorrento, Italy, May 1994.

[22] T.E. Flick, L.K. Jones, R.G. Priest, and C. Herman. Pattern classification using projection pursuit. *Pattern Recognition*, 23(12):1367–1376, 1990.

[23] M. Frean. The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209, 1990.

[24] J.H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82(397):249–266, March 1987.

[25] J.H. Friedman. Adaptive spline networks. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 675–683. Morgan Kaufmann, San Mateo, CA, 1991.

[26] J.H. Friedman. Multivariate adaptive regression splines (with discussion). *The Annals of Statistics*, 19(1):1–141, 1991.

[27] J.H. Friedman. An overview of predictive learning and function approximation. In J.H. Friedman and H. Wechsler, editors, *From Statistics to Neural Networks. Theory and Pattern Recognition Applications*, ASI Proceedings, Subseries F. Springer-Verlag, 1994.

[28] J.H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76(376):817–823, 1981.

[29] B. Fritzke. Growing cell structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.

[30] O Fujita. Optimization of the hidden unit function in feedforward neural networks. *Neural Networks*, 5:755–764, 1992.

[31] K.I. Funahashi. On the approximate realization of continous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[32] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[33] F. Girosi. Some extensions of radial basis functions and their applications in artificial intelligence. *Computers & Mathematics with Applications*, 24(12):61–80, December 1992.

[34] L.O. Hall, A.M. Bensaid, L.P. Clarke, R.P. Velthuizen, M.S. Silbiger, and J.C. Bezdek. A comparison of neural network and fuzzy clustering techniques in segmenting magnetic resonance images of the brain. *IEEE Transactions on Neural Networks*, 3(5):672–682, September 1992.

[35] S.J. Hanson and L.Y. Pratt. Comparing biases for minimal network construction with back-propagation. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 177–185. Morgan Kaufmann, San Mateo, CA, 1989.

[36] E. Hartman, J. Keeler, and J. Kowalski. Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation*, 2:210–215, 1990.

[37] B. Hassibi and D.G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*, pages 164–171. Morgan Kaufmann, San Mateo, CA, 1993.

[38] T.J. Hastie and R.J. Tibshirani. *Generalized Additive Models*. Monographs on Statistics and Applied Probability 43. Chapman and Hall, 1st edition, 1990.

[39] Y. Hirose, K. Yamashita, and S. Hijiya. Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, 4:61–66, 1991.

[40] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

[41] K. Hornik. Some new results on neural network approximation. *Neural Networks*, 6:1069–1072, 1993.

[42] P.J. Huber. Projection pursuit (with discussion). *The Annals of Statistics*, 13(2):435–525, 1985.

[43] J.N. Hwang, S.R. Lay, M. Maechler, D. Martin, and J. Schimert. Regression modeling in back-propagation and projection pursuit learning. *IEEE Transactions on Neural Networks*, 5(3):342–353, May 1994.

[44] T.T. Jervis and W.J. Fitzgerald. Optimization schemes for neural networks. CUED/F-INFENG/TR TR 144, Cambridge University Engineering Department, 1993.

[45] L.K. Jones. On a conjecture of Huber concerning the convergence of projection pursuit regression. *The Annals of Statistics*, 15(2):880–882, 1987.

[46] V. Kadirkamanathan and M. Niranjan. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5(6):954–975, 1993.

[47] E.D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, June 1990.

[48] N. Karunanithi and D. Whitley. Prediction of software reliability using feedforward and recurrent neural nets. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 800–805, Baltimore, MD, USA, June 1992.

[49] K. Khorasani and W. Weng. Structure adaptation in feed-forward neural networks. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 3, pages 1403–1408, Orlando, Florida, USA, June 1994.

[50] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[51] A.N. Kolmogorov and S.V. Fomin. *Introductory Real Analysis*. Dover, 1975.

[52] T.Y. Kwok and D.Y. Yeung. Improving the approximation and convergence capabilities of projection pursuit learning. *Neural Processing Letters*, 2(3), May 1995.

[53] T.Y. Kwok and D.Y. Yeung. Objective functions for training new hidden units in constructive neural networks, 1995. Submitted.

[54] S. Lay, J. Hwang, and S. You. Extensions to projection pursuit learning networks with parametric smoothers. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 3, pages 1325–1330, Orlando, Florida, USA, June 1994.

[55] Y. Le Cun, J.S. Denker, and S.A. Solla. Optimal brain damage. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan Kaufmann, San Mateo, CA, 1990.

[56] H. Lee, K. Mehrota, C.K. Mohan, and S. Ranka. An incremental network construction algorithm for approximating discontinuous functions. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 2191–2196, Orlando, Florida, USA, June 1994.

[57] S. Lee and R.M. Kil. A Gaussian potential function network with hierarchically self-organizing learning. *Neural Networks*, 4:207–224, 1991.

[58] E. Littmann and H. Ritter. Cascade LLM networks. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks*, volume 2, pages 253–257. Elsevier Science Publishers, 1992.

[59] E. Littmann and H. Ritter. Cascade network architectures. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 398–404, Baltimore, MD, USA, June 1992.

[60] J. Luo. A bias architecture with rank-expanding algorithm for neural networks supervised learning problem. In *Proceedings of the World Congress on Neural Networks*, volume 3, pages 742–747, San Diego, CA, June 1994.

[61] D.J.C. Mackay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, May 1992.

[62] M. Marchand, M. Golea, and P. Ruján. A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11(6):487–492, 1990.

[63] S.J. McKenna, I.W. Ricketts, A.Y. Cairns, and K.A. Hussein. Cascade-correlation neural networks for the classification of cervical cells. In *IEE Colloquium on Neural Networks for Image Processing Applications*, pages 5/1–4, London, UK, October 1992.

[64] G.F. Miller, P.M. Todd, and S.U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, 1989.

[65] T.M. Mitchell. The need for biases in learning generalizations. In J.W. Shavlik and T.G. Dietterich, editors, *Readings in Machine Learning*, pages 184–191. Morgan Kaufmann, 1990.

[66] J. Moody. Prediction risk and architecture selection for neural networks. In V. Cherkassky, J.H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, volume 136 of *NATO ASI Series F*, pages 147–165. Springer-Verlag, 1994.

[67] J. Moody and N. Yarvin. Networks with learned unit response functions. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 1048–1055. Morgan Kaufmann, San Mateo, CA, 1992.

[68] M.C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 107–115. Morgan Kaufmann, San Mateo, CA, 1989.

[69] T.M. Nabhan and A.Y. Zomaya. Toward generating neural network structures for function approximation. *Neural Networks*, 7(1):89–99, 1994.

[70] D.E. Nelson and S.K. Rogers. A taxonomy of neural network optimality. In *Proceedings of the IEEE National Aerospace and Electronics Conference*, volume 3, pages 894–899, Dayton, OH, USA, May 1992.

[71] J. Park and I. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3:246–257, 1991.

[72] J. Park and I.W. Sandberg. Approximation and radial-basis-function networks. *Neural Computation*, 5:305–316, 1993.

[73] R.E. Parker and M. Tummala. Identification of Volterra systems with a polynomial neural network. In *Proceedings of the 1992 IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 4, pages 561–564, San Francisco, CA, USA, March 1992.

[74] D.S. Phatak and I. Koren. Connectivity and performance tradeoffs in the cascade correlation learning architecture. *IEEE Transactions on Neural Networks*, 5(6):930–935, November 1994.

[75] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3:213–225, 1991.

[76] R. Reed. Pruning algorithms - a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, September 1993.

[77] B.D. Ripley. Statistical aspects of neural networks. In O.E. Barndorff-Nielsen, J.L. Jensen, and W.S. Kendall, editors, *Networks and Chaos – Statistical and Probabilistic Aspects*, pages 40–123. Chapman and Hall, London, 1993.

[78] B.D. Ripley. Neural networks and related methods for classification (with discussion). *Journal of the Royal Statistical Society Series B*, 56, 1994.

[79] J. Rissanen. Modelling by shortest data description. *Automatica*, 14:465–471, 1975.

[80] S. Roberts and L. Tarassenko. A probabilistic resource allocating network for novelty detection. *Neural Computation*, 6(2):270–284, March 1994.

[81] C.B. Roosen and T.J. Hastie. Logistic response projection pursuit. Technical Report BL011214-930806-09TM, AT&T Bell Laboratories, August 1993.

[82] C.B. Roosen and T.J. Hastie. Automatic smoothing spline projection pursuit. *Journal of Computational and Graphical Statistics*, 3(3):235–248, 1994.

[83] A. Saha, C.L. Wu, and D.S. Tang. Approximation, dimension reduction, and non-convex optimization using linear superpositions of Gaussians. *IEEE Transactions on Computers*, 42(10):1222–1233, October 1993.

[84] W.S. Sarle. Neural network implementation in sas software. In *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, 1994.

[85] W.S. Sarle. Neural networks and statistical models. In *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, April 1994.

[86] R. Setiono and L.C.K. Hui. Use of a quasi-Newton method in a feedforward neural network construction algorithm. *IEEE Transactions on Neural Networks*, 6(1):273–277, 1995.

[87] Y. Shin and J. Ghosh. The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 13–18, Seattle, Washington, July 1991.

[88] Y. Shin and J. Ghosh. Ridge polynomial networks. *IEEE Transactions on Neural Networks*, 6(2), May 1995.

[89] T.R. Shultz and W.C. Schmidt. A cascade-correlation model of balance scale phenomena. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 635–640, Hillsdale, NJ, 1991.

[90] N. Simon, H. Corporaal, and E. Kerckhoffs. Variations on the cascade-correlation learning architecture for fast convergence in robot control. In *Proceedings of the Fifth International Conference on Neural Networks and their Applications*, pages 455–464, Nimes, France, November 1992.

[91] S. Sjϕgaard. Generalization in cascade-correlation networks. In *Neural Networks for Signal Processing II. Proceedings of the IEEE - SP Workshop*, pages 59–68, Helsingoer, Denmark, September 1992.

[92] I.G. Smotroff, D.H. Friedman, and D. Connolly. Self organizing modular neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, Seattle, WA, USA, July 1991.

[93] P.L. Springer and S. Gulati. Parallelizing the cascade-correlation algorithm using Time Warp. *Neural Networks*, 8(4):571–577, 1995.

[94] C.S. Squires and J.W. Shavlik. Experimental analysis of aspects of the cascade-correlation learning architecture. Machine Learning Research Group Working Paper 91-1, Computer Sciences Department, University of Wisconsin-Madison, 1210 West Dayton Street, Madison, WI 53706, USA, 1991.

[95] Statistical Sciences, Inc., Seattle, Washington. *S-Plus User's Manual*, 3.0 edition, September 1991.

[96] C.C. Teng and B.W. Wah. An automated design system for finding the minimal configuration of a feed-forward neural network. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 3, pages 1295–1300, Orlando, Florida, USA, June 1994.

[97] M.F. Tenorio and W.T. Lee. Self-organizing network for optimum supervised learning. *IEEE Transactions on Neural Networks*, 1(1):100–110, March 1990.

[98] W. Verkooijen and H. Daniels. Connectionist projection pursuit regression. *Computational Economics*, 7:155–161, 1994.

[99] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman. Generalization by weight-elimination with application to forecasting. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 875–882. Morgan Kaufmann, San Mateo, CA, 1991.

[100] G. Weiss. Neural networks and evoluntionary computation. Part 1: Hybrid approaches in artificial intelligence. In *Proceedings of the First IEEE Conference on Evoluntionary Computation*, volume 1, pages 268–272, Orlando, FL, USA, June 1994.

[101] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14:347–361, 1990.

[102] L. Xu, S. Klasa, and A. Yuille. Recent advances on techniques of static feedforward networks with supervised learning. *International Journal of Neural Systems*, 3(3):253–290, 1992.

[103] X. Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 8:539–567, 1993.

[104] D.Y. Yeung. Constructive neural networks as estimators of Bayesian discriminant functions. *Pattern Recognition*, 26(1):189–204, 1993.

[105] J.L. Yuan and T.L. Fine. Forecasting demand for electric power. In B. Hassibi and D.G. Stork, editors, *Advances in Neural Information Processing Systems 5*, pages 739–746. Morgan Kaufmann, San Mateo, CA, 1993.

[106] B.T. Zhang. An incremental learning algorithm that optimizes network size and sample size in one trial. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 215–220, Orlando, Florida, USA, June 1994.