Objective Functions for Training New Hidden Units in Constructive Neural Networks

Tin-Yau Kwok and Dit-Yan Yeung, *Member, IEEE*

Abstract— **In this paper, we study a number of objective functions for training new hidden units in constructive algorithms for multilayer feedforward networks. The aim is to derive a class of objective functions the computation of which and the corresponding weight updates can be done in $O(N)$ time, where N is the number of training patterns. Moreover, even though input weight freezing is applied during the process for computational efficiency, the convergence property of the constructive algorithms using these objective functions is still preserved. We also propose a few computational tricks that can be used to improve the optimization of the objective functions under practical situations. Their relative performance in a set of two-dimensional regression problems is also discussed.**

Keywords— **Constructive algorithms, cascade-correlation, convergence, input weight freezing, quickprop.**

I. 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 feedforward networks is perhaps the most popular. Standard back-propagation performs gradient descent only in the weight space of a network with fixed topology. In general, it is 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. This can be easily understood by analogy to the problem of curve fitting using polynomials. Consider a data set generated from a smooth underlying function with additive noise on the outputs. A polynomial with too few coefficients will be unable to capture the underlying function from which the data was generated, while a polynomial with too many coefficients will fit the noise in the data and again result in a poor representation of the underlying function. For an optimal number of coefficients, the fitted polynomial will give the best representation of the function and also the best predictions for new data. A similar situation arises in the application of neural networks, where it is again necessary to match the network complexity to the problem being solved. Algorithms that can find an appropriate network architecture automatically are thus highly desirable.

A. Matching the Network Complexity to the Problem

There are three major approaches to tackle this problem. The first involves using a larger than needed network and training it until an acceptable solution is found. After this, some hidden units or weights are removed if they are no longer actively used. Methods using this approach are called *pruning* algorithms. The second approach, which corresponds to *constructive* algorithms, starts with a small network and then grows additional hidden units and weights until a satisfactory solution is found. Review for pruning algorithms can be found in [1], while that for constructive algorithms in [2], [3], [4].

The constructive approach has a number of advantages over the pruning approach. Firstly, for constructive algorithms, it is straightforward to specify an initial network[1], whereas for pruning algorithms, one does not know in practice how big the initial network should be. Secondly, constructive algorithms always search for small network solutions first. They are thus more computationally economical than pruning algorithms, in which the majority of the training time is spent on networks larger than necessary. Thirdly, as many networks with different sizes may be capable of implementing acceptable solutions, constructive algorithms are likely to find smaller network solutions than pruning algorithms. Smaller networks are more efficient in forward computation and can be described by a simpler set of rules. Functions of individual hidden units may also be more easily visualized. Moreover, by searching for small networks, the amount of training data required for good generalization may be reduced. Fourthly, pruning algorithms usually measure the change in error when a hidden unit or weight in the network is removed. However, such changes can only be approximated for computational efficiency, and hence may introduce large errors, especially when many are to be pruned.

The third approach is *regularization* [5], [6], [7]. However, regularization cannot alter the network structure, which must be specified in advance by the user. Although in principle the use of regularization should allow the network to be overly large, in practice using an overly large network makes optimization of the regularized error function with respect to its network weights more computationally intensive and difficult. Moreover, there is a delicate balance, controlled by a regularization parameter, between the error term and the penalty term. Early attempts either set this regularization parameter manually [5], [6] or by *ad hoc* procedures [7]. A more disciplined and long respected method is cross-validation [8]. However, this approach is usually very slow for nonlinear models like neural networks, because a large number of nonlinear optimization problems must be repeated. Recently, several researchers [9], [10], [11], [12] have incorporated Bayesian methods into

The authors are with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Email: {jamesk,dyyeung}@cs.ust.hk .

[1]The smallest possible network to start with has no hidden units. If prior knowledge of the problem is available, an alternative initial state may be supplied by the user.

neural network learning. Regularization can then be accomplished by using appropriate priors that favor small network weights (such as the normal [10] or Laplace [13] distribution), and the regularization parameter can be automatically set. However, Bayesian inference mechanisms must often assume asymptotic normality of the posterior distributions, which may break down when the number of weights in the network is large compared to the training set size[2]. This scenario is more likely to occur in overly large networks, further escalating the problem of determining an appropriate initial network for use in regularization.

Hence, in this paper, we will focus mainly on constructive algorithms. Moreover, we will be particularly interested in regression problems, in which networks with linear output units are the most common. Note that classification problems can be considered as a special case of regression problems. Some constructive methods, such as [15], [16], [17], [18], that can *only* be applied to classification problems will not be discussed here.

B. How to Train New Hidden Units

There are three major problems involved in the design of constructive algorithms:

1. **How to connect**: How to connect the new hidden unit to the existing network?
2. **How to train**: How to determine the (new and existing) weights in the network after connections to the new hidden unit are made?
3. **When to stop**: When to stop the addition of new hidden units, or, in other words, what is the optimal number of hidden units to be installed in the network?

The first issue has direct implications to the resultant network architecture. It is well-known that no single class of network architectures is ideal for all problems [19], and so different connection strategies for the new hidden units are suitable for different problems. But a comprehensive comparison of different architectures is still lacking.

The third issue on model selection may be addressed using different techniques, such as by comparing evidence in a Bayesian framework [9], [14], or, in a non-Bayesian framework, by the use of some information criteria [20], [21], [22], [23], [24], [25] or data resampling methods [26], [27], [28]. This is still an active research topic. Interested readers may see the discussions in [26], [29], [30].

In this paper, we will focus on the second issue. In principle, it is fairly straightforward as one can simply train the whole network again (as in [31], [32]). However, while in neural networks with fixed architecture, the weights are trained only once, here in constructive algorithms, training must be repeated every time hidden units are added. Hence, computational efficiency, in terms of both time and space, becomes an important issue. Most optimization routines do not scale up well when the number of weights k is large. For example, Newton's method requires computation of the Hessian matrix, entailing a space requirement of $O(k^2)$ and a time requirement of $O(k^3)$ in each iteration. Quasi-Newton methods still have a space requirement of $O(k^2)$, but do not require evaluating the Hessian matrix and are implementable in ways which only require $O(k^2)$ arithmetic operations in each iteration. The price to pay is that instead of having the quadratic local convergence of Newton's method, quasi-Newton methods only exhibit a local superlinear convergence rate. Conjugate gradient methods do not require storage or maintenance of the Hessian, but their local convergence rate is roughly linear only. Simple gradient descent methods, though they only require $O(k)$ space and time for each iteration, are notoriously slow when the network size is large [33]. Thus, in short, techniques to improve computational efficiency are essential in practical constructive algorithms, and these will be discussed in Section I-B.1.

On the other hand, a frequently overlooked issue concerning constructive algorithms is their convergence properties. The question of convergence can be stated as follows: Can the constructive algorithm produce a convergent sequence of network functions $\{f_n\}$ that, in the limit, approximates the target function f as closely as desired? In other words, does the sequence $\{f_n\}$ so generated strongly converge[3] to f as $n \to \infty$? Apparently, the universal approximation capability[4] of a network structure is necessary for the convergence of its constructive algorithm. However, we will see in Section II why this may not be sufficient.

B.1 Improving Computational Complexity

A common technique used in constructive algorithms (like [39], [40], [41], [42], [43]) to simplify the optimization problem and thus improve computational complexity is to assume that the hidden units already existing in the network are useful in modeling part of the target function. Hence, we can keep the weights feeding into these hidden units fixed (*input weight freezing*), and allow only the weights connected to the new hidden unit and the output units to vary. The number of weights to be optimized, and the time and space requirements for each iteration, can thus be greatly reduced. This reduction is especially significant when there are already many hidden units installed in the network.

Training of the adaptable weights may then be performed by back-propagation as usual [42], but the computational requirement may be further reduced 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 the hidden units to the outputs are trained (*output training*). In this way, only one layer of weights needs to be optimized each time. There is never any need to back-propagate the error

[2]As an example, MacKay [14] reported that the Gaussian approximation in the evidence framework seemed to break down significantly for $N/k < 3 \pm 1$, where N is the number of training patterns and k is the number of weights in the network.

[3]A sequence $\{f_n\}$ *strongly converges* [34] to f if $\lim_{n\to\infty} \|f - f_n\| = 0$, where $\|\cdot\|$ is the norm for the function space being considered.

[4]A class of networks is capable of *universal approximation* if it is broad enough to contain the target function f or a good enough approximation of f. See, for example, [35], [36] and [37], [38] for universal approximation results on multi-layer perceptrons and radial basis function networks respectively.

signals and hence is much faster.

During input training, the weights feeding into the new hidden unit are trained to optimize an objective function. The following objective functions have been used:

1. Projection index [44] that finds "interesting" projections deviating from the Gaussian form. However, this is probably more suitable for exploratory data analysis [45] rather than for regression or classification problems, as projections of the Gaussian form may also be useful as feature detectors.
2. The same error criterion, such as the squared error criterion, as used in output training [40]. In this method, the weights in each layer are updated in turn, by keeping all the other weights unchanged, and the whole process is cycled many times. However, it will be shown later in this paper that such a criterion is inferior to the others proposed in Section II.
3. The covariance between the residual error and the new hidden unit activation, as in the cascade-correlation architecture [39] and its variants. Without loss of generality, we assume that there is only one output unit in the network. New hidden units are added one at a time in a greedy manner by maximizing:

$$S_{cascor} = |\sum_{p}(E_p - \bar{E})(H_p - \bar{H})|, \qquad (1)$$

where p ranges over the training patterns, H_p is the activation of the new hidden unit for pattern p, E_p is the residual error for pattern p before the new hidden unit is added, and $\bar{H}$ and $\bar{E}$ are the corresponding values averaged over all patterns. Note that if there are N training patterns, then the time complexities for computing (1) and its weight update are both of $O(N)$ only. However, the design of S_{cascor} is rather *ad hoc*. As mentioned in [39], early versions of the architecture used a true correlation measure, but the version of S_{cascor} in (1) works better in most situations.
4. An objective function based on the use of projection matrices [46]. This can be formally derived, but both the computation of this objective function and each weight update require $O(N^2)$ multiplications. The space requirement is also $O(N^2)$. Ideally, this N, the number of training patterns, should be large for good generalization performance, and should also scale up with problem complexity [47]. Hence, this objective function may soon become infeasible for larger-scale problems. Moreover, its complex form makes it vulnerable to the problem of local optima in the optimization process.
5. Another very *ad hoc* objective function proposed in [48].

B.2 Effect on the Convergence Property

Input weight freezing, though highly beneficial in improving computational efficiency, unfortunately also affects the convergence property of the constructive algorithm. Consider, without loss of generality, feedforward networks with one hidden layer, whose network functions can be described by the set

$$\mathcal{F} = \bigcup_{n=1}^{\infty} \mathcal{F}_n,$$

where

$$\mathcal{F}_n = \{f_n \mid f_n(x) = \sum_{j=1}^{n} \beta_j \gamma(a_j^T x + \theta_j), \; a_j \in \Re^d, \beta_j, \theta_j \in \Re\},$$

with a_j being the weights connecting the inputs to the jth hidden unit, β_j the weight connecting this hidden unit to the output, and γ the hidden unit transfer function. In a certain function space (such as the L^p space), universal approximation results state that when γ satisfies certain mild conditions (e.g., bounded and non-constant [49]), $\mathcal{F}$ will be dense in that space, i.e., for any given target function f in the space, there exists a sequence of network functions $\{f_n\}$ in $\mathcal{F}$ such that $\{f_n\}$ strongly converges to f. Notice that under this formulation, all parameters (weights) in any f_n in the sequence are freely adjustable. However, when input weight freezing is used, the new network estimate f_n, constructed from f_{n-1}, cannot change the input weights of the hidden units associated with f_{n-1}. This restriction may then impede the approximation capabilities of the sequence $\{f_n\}$. A major aim of this paper is to find objective functions such that the convergence property will be preserved when these functions are used in the constructive algorithms. Recently, there are also some results on the convergence issue. We will postpone the discussion of them to Section III.

Also, note that the norm used in the convergence definition should be based on the whole input space, not just on the training patterns as is done in [48], [50]. 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 [48], [50], is inadequate.

C. Organization of this Paper

Motivated by the success of the cascade-correlation architecture in a variety of problems [51], [52], [53], [54], [55], the obscure design of the objective function and the importance of having convergence properties for constructive algorithms, we discuss in Section II a class of objective functions for training new hidden units in constructive algorithms, all of which have a time complexity of $O(N)$ without compromising the convergence property. A comparison with related works on the issue of convergence will be discussed in Section III. Practical problems in optimizing the objective functions, together with some suggested remedies, will be discussed in Section IV. Experimental comparison in using these objective functions for a series of regression problems will be described in Section V, followed by some concluding remarks in the last section. Proofs of

the mathematical results and extension to the case of nonlinear output units are in the Appendix.

II. Design of the Objective Functions

In the following, we assume that the network has only one linear output unit. The case for nonlinear output units is dealt with in Appendix B. Extension to multiple output units is straightforward. Moreover, we will focus on the L^2 space, i.e. the space of all square integrable functions. For $u, v \in L^2$, the inner product[5] $\langle u, v\rangle$ is defined by

$$\langle u, v\rangle = \int_X u(x)v(x)d\mu(x),$$

where μ is the (positive) input environment measure, X is the whole input space which is a bounded measurable subset in the d-dimensional Euclidean space $\Re^d$. The norm in L^2 space will be denoted as $\|\cdot\|$.

A network, having $n-1$ hidden units directly connected to the output unit, implements the function f_{n-1} given by:

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

where g_j represents the function implemented by the jth hidden unit. Note that these g_j's may only be indirectly connected to the input units through intermediate hidden units, and thus the network is not restricted to having only one single hidden layer. Moreover, $e_{n-1} \equiv f - f_{n-1}$ is the residual error function for the current network with $n-1$ hidden units.

Addition of a new hidden unit proceeds in two steps:

1. Input training: Find β_n and g_n such that the resultant linear combination of g_n with the current network, i.e. $f_{n-1} + \beta_n g_n$, gives minimum residual error $\|e_n\|$.
2. Output training: Keeping $g_1, g_2, \ldots, g_n$ fixed, adjust the values of $\beta_1, \beta_2, \ldots, \beta_n$ so as to minimize the residual error.

Note that, as will be shown later, β_n can be found as a by-product in step 1, without requiring back-propagation of error. This β_n can then be used as an initial value in performing step 2.

Step 1 is performed by maximizing an objective function over g_n from a set Γ. This Γ, having finite or infinite number of elements, contains all hidden unit functions that can possibly be implemented. It thus depends on the particular constructive algorithm that specifies how the new hidden unit is connected to the existing network, how the net input to the hidden unit is computed, and the transfer function of the new hidden unit, etc. Step 2 is used to ensure that $f-f_n$ remains orthogonal to the subspace spanned by $g_1, g_2, \ldots, g_n$. This minimization can be performed by using gradient descent or, when the output unit is linear and squared error criterion is used, by computing the pseudo-inverse.

A. Derivation for S_1 and S_2

We first address the problem in step 1.

Proposition 1: For a fixed $g \in \Gamma$ ($\|g\| \neq 0$)[6], the expression $\|f - (f_{n-1} + \beta g)\|$ achieves its minimum iff

$$\beta = \beta^* = \frac{\langle e_{n-1}, g\rangle}{\|g\|^2}. \tag{2}$$

Moreover, with β_n^* and β^* as defined in (2), $\|f - (f_{n-1} + \beta_n^* g_n)\| \leq \|f - (f_{n-1} + \beta^* g)\| \ \ \forall g \in \Gamma$, iff

$$\frac{\langle e_{n-1}, g_n\rangle^2}{\|g_n\|^2} \geq \frac{\langle e_{n-1}, g\rangle^2}{\|g\|^2} \ \ \forall g \in \Gamma.$$

(Proofs are in Appendix A.)

Thus, proposition 1 suggests that the objective function to be optimized during input training is:

$$\frac{\langle e_{n-1}, g_n\rangle^2}{\|g_n\|^2}. \tag{3}$$

However, in practice, (3) can only be calculated when the exact functional form of e_{n-1} is available, which is obviously impossible as the true f is unknown. A consistent[7] estimate of (3) using information from the training set is $\frac{(\frac{1}{N}\sum_p E_p H_p)^2}{\frac{1}{N}\sum_p H_p^2}$, where H_p is the activation of the new hidden unit for pattern p and E_p is the corresponding residual error before this new hidden unit is added. Dropping the factor $\frac{1}{N}$ which is common for all candidate hidden unit functions, we obtain the following objective function:

$$S_1 = \frac{(\sum_p E_p H_p)^2}{\sum_p H_p^2}. \tag{4}$$

The corresponding update equation for connection weight w_i feeding into this new hidden unit is:

$$\begin{aligned}\Delta w_i \ &\propto \ \frac{\partial S_1}{\partial w_i} \\ &\propto \ \left[\sum_p H_p^2 \cdot \sum_p E_p \frac{\partial H_p}{\partial w_i} - \sum_p E_p H_p \cdot \sum_p H_p \frac{\partial H_p}{\partial w_i}\right] \\ &\quad \cdot \left(\sum_p E_p H_p\right) / \left(\sum_p H_p^2\right)^2.\end{aligned}$$

Notice that the time complexities in computing this objective function and each weight update are of $O(N)$. Moreover, storage requirement is minimal as the E_p's and H_p's need not be stored.

[5] An *inner product* [34] in a real linear space R is a real function defined for every pair of elements $u, v \in R$ and is denoted by $\langle u, v\rangle$, with the following properties:
1. $\langle u, u\rangle \geq 0$ where $\langle u, u\rangle = 0$ if and only if $u = 0$;
2. $\langle u, v\rangle = \langle v, u\rangle$;
3. $\langle \lambda u, v\rangle = \lambda\langle u, v\rangle, \ \ \lambda \in \Re$;
4. $\langle u, v + w\rangle = \langle u, v\rangle + \langle u, w\rangle$.

[6] $\|g\| = 0$ implies $g(x) \equiv 0 \ \ \forall x \in X$, which should obviously be excluded.

[7] Consistency follows as the estimate is a continuous function of the sample product moments.

The weight β_n connecting the new hidden unit to the output unit is also determined as a by-product, given by $\beta_n = \sum_p E_p H_p / \sum_p H_p^2$ as determined in (2). Compared to [40] in which the same criterion is used for both the input and output training phases, they have to (randomly) fix an initial guess for β_n before input training can proceed. Hence, the hidden unit found is likely to be suboptimal, and so the input and output training phases have to be iterated for several times, significantly lengthening the optimization process. In our approach, however, the optimal β_n for the new hidden unit is automatically determined and its value is not required during input training. Moreover, the weight update process in [40] can be regarded as a variant of the coordinate descent method [56], which is known to have poor convergence properties. Simulation results in Section V also confirm that this method used without iteration is inferior.

The convergence property of the constructive algorithm using (3) as the objective function during input training, with input weight freezing in effect, is assured by the following theorem:

Theorem 1: Given span(Γ) is dense in L^2 and $\forall g \in \Gamma$, $0 < \|g\| < b$ for some $b \in \Re$. If g_n is selected as to maximize $\langle e_{n-1}, g\rangle^2 / \|g\|^2$, then $\lim_{n\to\infty} \|f - f_n\| = 0$.

Requiring the set of hidden unit functions to satisfy the denseness requirement is not hard, as is supported by various universal approximation results for feedforward networks [36], [38], [49], [57]. Moreover, the boundedness assumption of $\|g\|$ holds for continuous transfer functions on bounded domains. Hence, Theorem 1 shows that the sequence of networks so constructed incrementally strongly converges to the target function.

If we further restrict the target function f to be an *exact* summation of basis functions from Γ, i.e. of the form

$$f = \sum_{j=1}^{m} v_j g_j^* \tag{5}$$

for some $g_j^* \in \Gamma$, then, similar to [58], we can obtain the following convergence rate:

Proposition 2: If the target function is an exact summation of basis functions from Γ as in (5) and the network is constructed as in Theorem 1, then

$$\|e_n\|^2 < \frac{b_f^2 \|e_0\|^2}{n\|e_0\|^2 + b_f^2}, \tag{6}$$

where $b_f \equiv m \max_{j=1,\ldots,m} |v_j| \|g_j^*\|$. Or, in other words, $\|e_n\|^2 = O(\frac{1}{n})$.

Because of the boundedness assumption of $\|g\|$, the denominator in (3) can be dropped without affecting convergence.

Corollary 1: With the conditions in Theorem 1, if g_n is selected as to maximize $\langle e_{n-1}, g\rangle^2$, then $\lim_{n\to\infty} \|f - f_n\| = 0$.

And we arrive at the second objective function:

$$S_2 = (\sum_p E_p H_p)^2. \tag{7}$$

Again, the time complexities in computing this objective function and each weight update are of $O(N)$.

If we restrict the target function to be an exact summation of basis functions from Γ, then

Proposition 3: If the target function is of the form in (5), $0 < \|g\| < b \ \ \forall g \in \Gamma$, and the network is constructed as in Corollary 1, then

$$\|e_n\|^2 < \frac{\bar{b}_f^2 \|e_0\|^2}{n\|e_0\|^2 + \bar{b}_f^2}, \tag{8}$$

where $\bar{b}_f \equiv bm \max_{j=1,\ldots,m} |v_j|$.

Note that

$$b_f \equiv m \max_{j=1,\ldots,m} |v_j| \|g_j^*\| \le bm \max_{j=1,\ldots,m} |v_j| \equiv \bar{b}_f,$$

which implies

$$\frac{b_f^2 \|e_0\|^2}{n\|e_0\|^2 + b_f^2} \le \frac{\bar{b}_f^2 \|e_0\|^2}{n\|e_0\|^2 + \bar{b}_f^2},$$

i.e. the bound for $\|e_n\|^2$ in (6) is smaller than that in (8). Of course, this does not necessarily mean that the network sequence $\{f_n\}$ constructed using S_1 as the objective function will converge faster than that using S_2, as it also depends on how tight the bounds in (6) and (8) are. Nevertheless, empirical results in Section V tend to support this difference in convergence rates.

B. Derivation for S_{cascor} and S_3

The objective function S_{cascor} used in the cascade-correlation architecture can also be derived from the above results. But let us first introduce a few notations.

Let $\mathcal{H} \equiv L^1 \cap L^2$. For $u, v \in \mathcal{H}$, define

$$\langle u, v\rangle = \int_X (u(x) - \bar{u}(x))(v(x) - \bar{v}(x))\, d\mu(x), \tag{9}$$

where

$$\bar{u}(x) = \frac{1}{\mu(X)} \int_X u(x)\, d\mu(x), \ \ \bar{v}(x) = \frac{1}{\mu(X)} \int_X v(x)\, d\mu(x).$$

Obviously, $\langle u, v\rangle$ is a semi inner product[8] in $\mathcal{H}$. Here we use the notation $\langle \cdot, \cdot \rangle$ for various spaces, which should be clear from context. The corresponding semi-norm is also denoted as $\|\cdot\|$.

To convert this semi inner product into an inner product, consider the set J of all constant-valued functions in $\mathcal{H}$. The *quotient space* $\mathcal{H}/J$ is the set with elements

$$\phi(u) = u + J = \{u + w : w \in J\},$$

for each $u \in \mathcal{H}$. For $\phi(u), \phi(v) \in \mathcal{H}/J$, define

$$\langle \phi(u), \phi(v)\rangle = \langle u, v\rangle, \tag{10}$$

where $\langle u, v\rangle$ is as defined in (9). Obviously, $\langle \phi(u), \phi(v)\rangle$ defines an inner product in $\mathcal{H}/J$. We then have the following corollary.

[8] A *semi inner product* differs from an inner product in that $\langle u, u\rangle$ may be 0 even when $u \neq 0$.

Corollary 2: With the conditions in Theorem 1, if g_n is selected as to maximize

$$\frac{\langle \phi(e_{n-1}), \phi(g_n) \rangle^2}{\|\phi(g_n)\|^2},$$

then $\lim_{n\to\infty} \|\phi(f) - \phi(f_n)\| = 0$.

Hence, as f and f_n differ by at most a constant when $n \to \infty$, $f_n \to f$ if a constant (bias) term is included in the expansion of f_n. The corresponding objective function to be maximized is:

$$S_3 = \frac{(\sum_p (E_p - \bar{E})(H_p - \bar{H}))^2}{\sum_p (H_p - \bar{H})^2}, \tag{11}$$

where $\bar{H}$ is the average value of the new hidden unit over all training patterns, and $\bar{E}$ is the corresponding average for the residual error. Again, only $O(N)$ time is required for computing (11) and its weight update.

Using Corollary 1, we can drop the denominator in (11) and arrive at the fourth objective function:

$$\left(\sum_p (E_p - \bar{E})(H_p - \bar{H}) \right)^2,$$

which is the square of S_{cascor}. Again, when the target function is restricted to be an exact summation of basis functions from Γ, the convergence rates for S_3 or S_{cascor} as objective functions are $O(\frac{1}{n})$.

Recall that the objective functions S_1 and S_2 are derived by assuming that f_{n-1} is fixed at each step when the new hidden unit g_n is optimized. In deriving S_3 and S_{cascor} by using the quotient space, however, we have basically used the knowledge that the bias to the output unit will be corrected during output training, and hence the mean values of E_p and H_p may be ignored during input training. This is sometimes beneficial, as is illustrated by the simulation results in Section V.

III. Related Work On the Convergence Issue

The convergence issue of constructive algorithms has been discussed in [59], [60], originally in the context of projection pursuit regression [60], [61]. When the norms of the elements g in Γ are bounded (i.e. $\|g\| < b$ as we also assumed in Theorem 1), convergence is shown in [59] when the target function f is in the closure of the convex hull of Γ. However, when the norm is bounded, the convex closure is no longer equal to the span of Γ, and thus convergence to all L^2 functions is not guaranteed. Whereas in our case, Theorem 1 holds for all L^2 functions.

Moreover, in both [59] and [60], the iterative sequence of network estimates is formed from a convex combination of the previous network function f_{n-1} and the new basis function g_n,

$$f_n = \alpha_n f_{n-1} + (1 - \alpha_n) g_n, \tag{12}$$

where $0 \le \alpha_n \le 1$. Whereas in our approach, the new f_n is formed from full linear combination of the old and new hidden unit functions. The weights connecting the old hidden units to the output unit are thus not constrained as a group, as in (12). Besides, in (12), α_n has to be learned together with the new hidden unit g_n, while in our case, the parameters of the new hidden unit are first learned and then the output layer weights. Training is thus performed for only one layer of weights at a time, making optimization of the objective function simpler. Moreover, the objective function to be minimized in [59], [60] is different from that developed in the previous section, and thus results in [59], [60] cannot be directly applied here.

On the other hand, while results in [59], [60] and those we studied here are in the context of the L^2 space (or more general Hilbert spaces), Darken *et al.* [62] extended the bounds on the rate of approximation to broader classes of spaces, including L^p spaces where $1 < p < \infty$. However, they are more interested in showing the *existence* of a convergent network sequence, while we are more interested in the convergence properties of specific methods that are able to deliver such sequences.

Drago and Ridella [58] also studied the convergence properties of using S_{cascor} as the objective function for function approximation, but only for the case when the hidden unit transfer function is the hyperbolic tangent function and uniform environment measure. Our work here poses little restriction on the input environment measure or the hidden unit transfer function, so long as the hidden unit transfer function is bounded and the requirements for universal approximation are met. Moreover, a number of other objective functions, besides S_{cascor}, are derived and discussed.

Recently, Kůrková and Beliczynski [63] also devised the S_2 criterion in this paper as an objective function. However, their convergence proof follows from that in [59], and hence is restricted to target functions that are in the convex closure of Γ.

IV. Some Practical Concerns

A. Practical Problems

Before discussing the simulation results, we first discuss a few practical problems in optimizing the objective functions.

A.1 Problem with the Objective Function

In the following, we denote any objective function obtained in the previous section simply by S. Its first and second partial derivatives with respect to the connection weight w_i will be denoted by S'_{w_i} and S''_{w_i} respectively.

As learning proceeds, hidden units are added to the network and the residual errors E_p's decrease with time. Observe that S, S'_{w_i} and S''_{w_i} are continuous with respect to E_p, and

$$S = S'_{w_i} = S''_{w_i} = 0,$$

when all the E_p's are zero. Hence,

$$S \to 0, \;\; S'_{w_i} \to 0, \;\; S''_{w_i} \to 0,$$

as $E_p \to 0 \;\; \forall p$. This may pose a problem when the residual errors are small but still not acceptable. If gradient-ascent

algorithms that use only first-order information (such as standard back-propagation) are used in the optimization, w_i is changed at each step by an amount

$$\Delta w_i = \eta S'_{w_i}, \tag{13}$$

where η is the learning rate. As S'_{w_i} is small, learning basically comes to a halt. By using optimization algorithms that use second-order information, we may be able to do a better job. Hence, in the experiments, the quickprop learning algorithm [33] is used to implement this optimization. However, the problem is still not solved completely.

A.2 Problem with the Quickprop Algorithm

The quickprop algorithm is a second-order method. It assumes that the function, S, to be optimized is locally quadratic with respect to each weight, and the Hessian matrix of S in weight space is diagonal. Although these assumptions are quite "risky", the technique works well in practice [33], [64].

Denoting S'_{w_i} at time t by $S'_{w_i}[t]$, the change for weight w_i at time t is given by:

$$\Delta w_i[t] = \frac{S'_{w_i}[t]}{S'_{w_i}[t-1] - S'_{w_i}[t]} \Delta w_i[t-1], \tag{14}$$

where $\Delta w_i[t-1]$ is the weight update at time $t-1$.

There are cases when weight update by (14) is not used. For example, since quickprop changes weights based on what happened during the previous weight update, (13) is used to compute the first step. Another situation is when the current slope with respect to w_i is in the same direction as that of the previous slope, and its magnitude is close to or even larger than the previous one. In this case, the next weight update is given by

$$\Delta w_i[t] = \mu \Delta w_i[t-1], \quad \mu > 1. \tag{15}$$

Although the aim of this restriction is to improve the stability of the algorithm, it may be problematic when one is exploring in a plateau[9] in the error surface. Consider the w_i direction in the weight space. Taking the gradient ascent step in (13) at $t = 0$,

$$w_i[1] = w_i[0] + \eta S'_{w_i}[0],$$

with

$$\Delta w_i[0] = \eta S'_{w_i}[0]. \tag{16}$$

Now, using Taylor series expansion and (16),

$$\begin{aligned} \frac{S'_{w_i}[1]}{S'_{w_i}[0]} &\simeq \frac{S'_{w_i}[0] + \Delta w_i[0] S''_{w_i}[0]}{S'_{w_i}[0]} \\ &= \frac{S'_{w_i}[0] + \eta S'_{w_i}[0] S''_{w_i}[0]}{S'_{w_i}[0]} \qquad (17) \\ &= 1 + \eta S''_{w_i}[0], \qquad (18) \end{aligned}$$

[9]A *plateau* is a region where the first and second derivatives of the function to be optimized with respect to all the parameters are nearly zero.

where $S''_{w_i}[t]$ denotes S''_{w_i} at time t. If $w_i[0]$ falls in the region of a plateau, then

$$S'_{w_i}[0] \simeq 0, \quad S''_{w_i}[0] \simeq 0.$$

(18) thus implies

$$\frac{S'_{w_i}[1]}{S'_{w_i}[0]} \simeq 1,$$

which satisfies the conditions for (15) to be applied. The next weight update, using (15) and (16), is then:

$$\Delta w_i[1] = \mu \eta S'_{w_i}[0] \simeq 0.$$

Movement in the weight space is thus very slow.

This problem, however, is usually not that severe. Although the next weight change is small, and even if the conditions for (15) to be applied continue to be satisfied, it can build up gradually given a sufficient number of training epochs. This can be seen clearly by applying (15) repeatedly,

$$\Delta w_i[t] = \mu^n \Delta w_i[t-n]. \tag{19}$$

Thus, as $\mu > 1$, the difference between $S'_{w_i}[t]$ and $S'_{w_i}[t-1]$ will finally be large enough for the quadratic approximation to be applied.

However, in constructive neural network algorithms, training is usually limited by a *patience* parameter [39], which means that the function to be optimized has to be improved by a certain fraction within a certain number of training epochs. Patience helps to end each learning phase when progress is excessively slow and thus saves time. Moreover, it is also experimentally shown to be able to improve generalization by avoiding overfitting of the training data [65]. However, the value of this patience parameter is usually small. Hence, waiting for the weight slopes to slowly build up as in (19) is not possible under such situations. More drastic action is needed to get out of the plateau in limited time. This situation is particularly acute in our case, as we have shown in the last section that when the residual errors are small, both the first and second derivatives of the objective function are likely to be small.

B. Remedies

To alleviate the problems mentioned above, we aim at increasing the slope of the objective function to be optimized when the residual errors are small, such that the region to be searched is less likely to be a plateau. On the other hand, if we are so unfortunate that quickprop really searches in a plateau, we aim at finding a better method to get out of it quickly.

To avoid the first problem, Fahlman [33] suggested to adjust S'_{w_i} by adding a small offset to the values of $\frac{\partial H_p}{\partial w_i}$. However, this distorts the true surface of S'_{w_i} and, as noted by Crowder [66], this "confuses the correlation machinery" and cannot solve the problem.

B.1 Transforming the Objective Function

To increase the slope of S when the residual errors are small, we transform S by a (possibly nonlinear) functional $t : \mathcal{C} \to \mathcal{C}$, where $\mathcal{C}$ is the space of all real-valued continuous functions:

$$\tilde{S} = t(S),$$

such that

$$\tilde{S}'_{w_i} \geq S'_{w_i}$$

when S is small. Moreover, we require the function $\hat{t} : \Re \to \Re$ in

$$(t(S))(\{E_p\}, \{H_p\}) = \hat{t}(S(\{E_p\}, \{H_p\})),$$

to be strictly increasing. This is desirable so that locations of the local and global optima of $\tilde{S}$ are the same as those of S.

An obvious choice of t is

$$t(S) = aS, \quad a > 1.$$

This amounts to scaling up the S dimension, or equivalently increasing the learning rate η. However, increasing η too much may cause oscillation, while increasing only a little may not be useful in improving the situation.

Another simple choice is

$$\tilde{S} = t(S) = \sqrt{S}.$$

Now $\tilde{S}'_{w_i} = \frac{S'_{w_i}}{2\sqrt{S}}$, and hence $\tilde{S}'_{w_i} > S'_{w_i}$ when $S < 0.25$. Thus, the slope is scaled up when S is small. The smaller S is, the larger is the scaling of the slope (Figure 1). Even when S is large, which makes $\tilde{S}'_{w_i}$ smaller than S'_{w_i}, it is not a problem because then quickprop will be mainly using the quadratic approximation (14).

Of course, there are other choices of t, such as

$$t(S) = S^{\frac{1}{k}}, \quad k > 1,$$

which basically changes the switch-over point where $\tilde{S}'_{w_i} > S'_{w_i}$. But the basic idea of all these schemes is to dynamically alter the slope of the objective function during learning. This is similar to the idea of adaptive learning rate in back-propagation [64], [67]. However, we demonstrate here that a simple change in the objective function to be optimized can achieve the same goal, without requiring modification to the learning algorithm or continual updating of the learning rate which incurs additional computational burden. Finally, notice that this kind of modification to the objective functions does not affect the convergence rates of the constructive algorithm, as given in (6) and (8).

B.2 Modification to the Quickprop Algorithm

For the second problem of enabling a faster escape from the plateau, a simple solution is to take large steps. From Section IV-A.2, we saw that the problem arises from always using the gradient ascent step when conditions to (15) are satisfied. To alleviate this, we take the quadratic approximation in (14) when the changes in all weight directions are very small, even under those conditions.

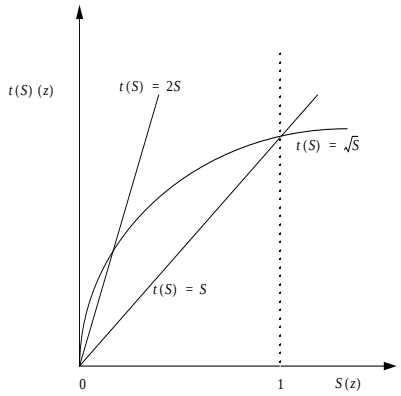


Fig. 1. Plot of several choices of f.

Following the analysis in Section IV-A.2, by taking the quadratic ascent step,

$$\begin{aligned}\Delta w_i[1] &= \frac{S'_{w_i}[1]}{S'_{w_i}[0] - S'_{w_i}[1]}\Delta w_i[0] \\ &\simeq \frac{\eta S'_{w_i}[0](S'_{w_i}[0] + \eta S'_{w_i}[0]S''_{w_i}[0])}{-\eta S'_{w_i}[0]S''_{w_i}[0]} \\ &= -S'_{w_i}[0]\left(\eta + \frac{1}{S''_{w_i}[0]}\right).\end{aligned}$$

It is simple to verify that when $|S''_{w_i}[0]| < \frac{1}{\eta(\mu+1)}$, as when $w_i[0]$ falls on a plateau, this will be of greater magnitude than the gradient ascent step.

V. Simulation

In this section, we compare the generalization performance of several networks constructed using the objective functions S_1 in (4), S_2 in (7), S_3 in (11), their modified versions (by taking the square root), S_{cascor} in (1), the objective function S_{fujita} used by Fujita in [46], and the squared error criterion[10] S_{sqr} used in [40]. Comparison is performed on a set of regression functions originally used in [40]. Preliminary study on the improvements in generalization performance by the modifications discussed in Section IV-B has also been reported in [68], in the context of chaotic time series prediction.

A. Setup

The networks under comparison all have linear output units and a single layer of hyperbolic tangent hidden units. Hidden units are added up to a maximum number of 15. Input training is performed by using the modified quickprop algorithm as described in Section IV-B.2, while output "training" is performed by computing the pseudo-inverse exactly.

The regression problems used are the two-dimensional functions which have been used in [40]. They are:

- Simple interaction function:

$$f^{(1)}(x_1, x_2) = 10.391((x_1 - 0.4)(x_2 - 0.6) + 0.36).$$

[10]Notice that unlike the other objective functions, S_{sqr} is always measured at the output nodes, and is minimized instead of being maximized during both the input and output training phases.

- Radial function:

$$f^{(2)}(x_1, x_2) = 24.234(r^2(0.75 - r^2)),$$
$$r^2 = (x_1 - 0.5)^2 + (x_2 - 0.5)^2.$$

- Harmonic function:

$$f^{(3)}(x_1, x_2) = 42.659((2 + x_1)/20 + \mathrm{Re}(z^5)),$$
$$z = x_1 + ix_2 - 0.5(1 + i),$$

or equivalently, with $\tilde{x}_1 = x_1 - 0.5, \tilde{x}_2 = x_2 - 0.5$,

$$f^{(3)}(x_1, x_2) = 42.659(0.1 + \tilde{x}_1(0.05 + \tilde{x}_1^4 - 10\tilde{x}_1^2\tilde{x}_2^2 + 5\tilde{x}_2^4)).$$

- Additive function:

$$f^{(4)}(x_1, x_2) = 1.3356(1.5(1 - x_1) + e^{2x_1 - 1}\sin(3\pi(x_1 - 0.6)^2) + e^{3(x_2 - 0.5)}\sin(4\pi(x_2 - 0.9)^2)).$$

- Complicated interaction function:

$$f^{(5)}(x_1, x_2) = 1.9(1.35 + e^{x_1}\sin(13(x_1 - 0.6)^2) e^{-x_2}\sin(7x_2)).$$

Plots of these functions are shown in Figures 2 to 6.

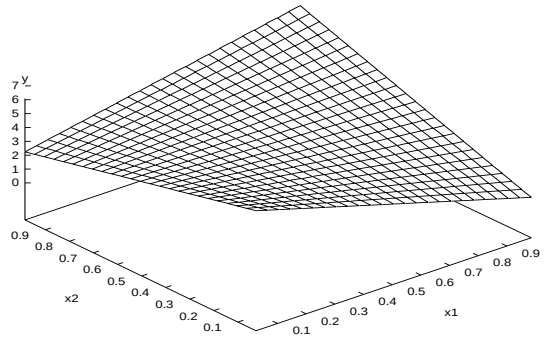


Fig. 2. $f^{(1)}$: simple interaction

We employ the same basic setup as in [40]. Two sets of training data, one noiseless and the other noisy, are generated. The noiseless training set has 225 points, and is generated from the uniform distribution $U[0, 1]^2$. The same set of abscissa values (x's) is used for experiments with all five functions. The test set, of size 10000, is generated from a regularly spaced grid on $[0, 1]^2$, and is also the same for all five functions. The noisy training set is generated by adding independent and identically distributed (i.i.d.) Gaussian noise, with mean zero and standard deviation 0.25, to the noiseless training set. Its size is thus also 225. Whereas results in [40] are based on only one specific set of training data, we want to get information on the variability due to the location of the x's. Hence, in the simulations below, we perform 100 independent trials each generating a different set of training data. The mean

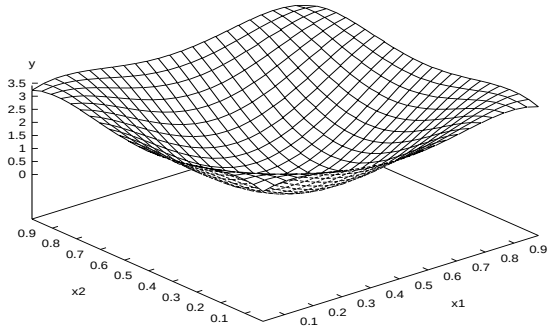


Fig. 3. $f^{(2)}$: radial

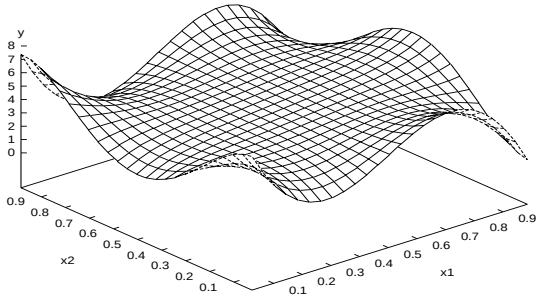


Fig. 4. $f^{(3)}$: harmonic

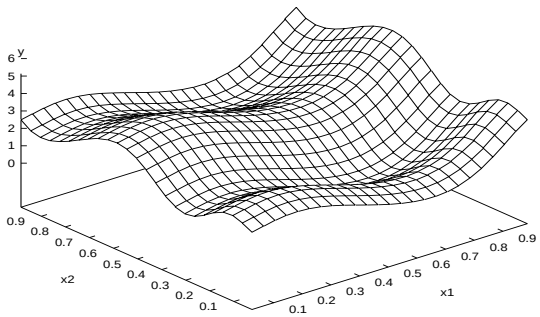


Fig. 5. $f^{(4)}$: additive

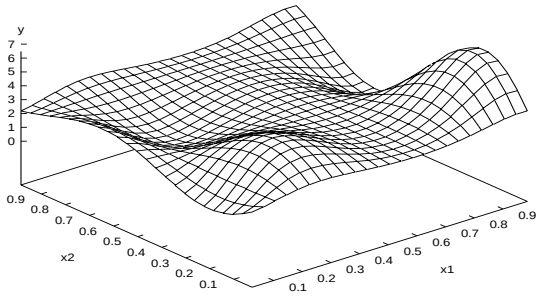


Fig. 6. $f^{(5)}$: complicated interaction

TABLE I
MEAN SIGNAL-TO-NOISE RATIOS FOR THE FIVE REGRESSION PROBLEMS.

	$f^{(1)}$	$f^{(2)}$	$f^{(3)}$	$f^{(4)}$	$f^{(5)}$
SNR	3.731	3.966	3.368	3.978	4.015

signal-to-noise ratios (SNR) for the five noisy functions are shown in Table I.

As in [40], the *fraction of variance unexplained* (FVU) on the test set is used for comparison. It is defined as:

$$FVU = \frac{\sum_{i=1}^{N}(f(x_i) - f_n(x_i))^2}{\sum_{i=1}^{N}(f(x_i) - \bar{f})^2},$$

where $\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$. Note that the FVU is proportional to the commonly used mean squared error. Moreover, as mentioned in Section I-B, in this paper we focus on how to determine the network weights after hidden unit addition, rather than on the issue of determining the optimal number of hidden units to be installed in the network. Hence, in the following, we assume that some perfect criterion has been used to handle this issue and we will use for comparison the lowest testing FVU among the 15 networks ranging from 1 to 15 hidden units in each trial.

B. Results

Boxplots for the best attainable testing (generalization) FVUs averaged over 100 independent trials are shown in Figures 8 and 9 in Appendix C. The corresponding means and medians are shown in Tables II and III. A pairwise comparison of the generalization performance for different objective functions, on a test by test basis, by using the sign test [69] at a 95% level of significance, is also reported in Tables IV to XIII. Each method (A, say) in the row is compared with each method (B, say) in the column. An asterisk in the corresponding entry indicates that the generalization performance of method A is *significantly* better than that of B. Boxplots are also shown in Figures 8 and 9 in Appendix C.

TABLE IV
NOISELESS $f^{(1)}$. HERE, s_c STANDS FOR S_{cascor}, s_f FOR S_{fujita} AND s_s FOR S_{sqr}.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*	*			*		
$\sqrt{s_1}$	*		*	*			*		
s_2									
$\sqrt{s_2}$			*						
s_3	*	*	*	*			*		*
$\sqrt{s_3}$	*	*	*	*			*		*
s_c			*						
s_f	*	*	*	*			*		*
s_s	*	*	*	*			*		

TABLE V
NOISELESS $f^{(2)}$.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*	*				*	*
$\sqrt{s_1}$	*		*	*				*	*
s_2									
$\sqrt{s_2}$			*					*	*
s_3	*	*	*	*			*	*	*
$\sqrt{s_3}$	*	*	*	*	*		*	*	*
s_c	*		*	*				*	*
s_f			*						
s_s			*						

TABLE VI
NOISELESS $f^{(3)}$.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*					*	*
$\sqrt{s_1}$	*		*	*			*	*	*
s_2									*
$\sqrt{s_2}$			*					*	*
s_3			*					*	*
$\sqrt{s_3}$			*					*	*
s_c			*					*	*
s_f			*						*
s_s									

TABLE VII
NOISELESS $f^{(4)}$.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*					*	*
$\sqrt{s_1}$			*					*	*
s_2									
$\sqrt{s_2}$	*	*	*					*	*
s_3	*	*	*	*			*	*	*
$\sqrt{s_3}$	*	*	*					*	*
s_c	*	*	*					*	*
s_f			*						*
s_s			*						

TABLE VIII
NOISELESS $f^{(5)}$.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*	*		*		*	*
$\sqrt{s_1}$	*		*	*	*	*	*	*	*
s_2									
$\sqrt{s_2}$			*					*	*
s_3			*			*		*	*
$\sqrt{s_3}$			*					*	*
s_c			*	*		*		*	*
s_f			*						*
s_s			*						

TABLE II

COMPARISON OF MEAN TESTING FVUS IN 100 TRIALS ON NOISELESS TRAINING SETS (MEDIANS ARE IN BRACKETS).

	S_1	$\sqrt{S_1}$	S_2	$\sqrt{S_2}$	S_3	$\sqrt{S_3}$	S_{cascor}	S_{fujita}	S_{sqr}
$f^{(1)}$	0.02096 (0.02049)	0.01117 (0.01006)	0.09528 (0.04882)	0.02435 (0.02242)	0.00293 (0.00234)	0.00258 (0.00204)	0.02483 (0.02335)	0.00369 (0.00250)	0.00693 (0.00506)
$f^{(2)}$	0.02917 (0.02925)	0.02799 (0.02746)	0.42609 (0.43811)	0.03083 (0.02994)	0.01972 (0.01819)	0.01818 (0.01702)	0.02668 (0.02626)	0.04712 (0.03845)	0.03829 (0.03461)
$f^{(3)}$	0.26904 (0.26449)	0.24738 (0.23830)	0.54700 (0.51089)	0.27514 (0.26555)	0.30569 (0.25403)	0.28812 (0.25964)	0.26503 (0.26219)	0.44422 (0.43758)	0.57347 (0.59375)
$f^{(4)}$	0.03555 (0.02898)	0.03680 (0.03241)	0.63554 (0.68748)	0.03055 (0.02710)	0.02710 (0.02224)	0.02959 (0.02633)	0.03054 (0.02702)	0.06977 (0.05821)	0.18468 (0.19432)
$f^{(5)}$	0.12088 (0.12022)	0.11126 (0.11180)	0.61000 (0.58597)	0.13404 (0.13158)	0.15967 (0.11616)	0.16739 (0.12822)	0.12114 (0.12078)	0.24607 (0.23344)	0.29447 (0.29685)

TABLE III

COMPARISON OF MEAN TESTING FVUS ON NOISY TRAINING SETS.

	S_1	$\sqrt{S_1}$	S_2	$\sqrt{S_2}$	S_3	$\sqrt{S_3}$	S_{cascor}	S_{fujita}	S_{sqr}
$f^{(1)}$	0.06980 (0.07985)	0.05940 (0.07181)	0.16909 (0.11176)	0.07067 (0.07977)	0.04582 (0.05640)	0.04432 (0.05421)	0.07092 (0.07903)	0.04836 (0.06050)	0.04969 (0.06160)
$f^{(2)}$	0.07347 (0.08150)	0.06951 (0.07943)	0.45520 (0.45535)	0.07275 (0.08169)	0.06564 (0.07621)	0.06426 (0.07445)	0.06915 (0.07876)	0.08587 (0.08348)	0.07997 (0.08298)
$f^{(3)}$	0.31637 (0.30799)	0.30770 (0.29644)	0.51487 (0.50011)	0.31702 (0.31273)	0.29254 (0.26530)	0.33864 (0.27951)	0.30941 (0.31104)	0.46683 (0.46308)	0.63408 (0.64252)
$f^{(4)}$	0.08546 (0.08953)	0.08453 (0.08834)	0.66685 (0.70169)	0.07980 (0.08605)	0.07820 (0.08705)	0.08010 (0.08704)	0.08302 (0.08589)	0.12513 (0.11231)	0.20317 (0.19736)
$f^{(5)}$	0.17353 (0.17164)	0.16175 (0.16872)	0.70805 (0.76016)	0.18254 (0.17416)	0.20077 (0.17426)	0.20082 (0.18430)	0.17613 (0.17336)	0.29649 (0.31078)	0.35143 (0.34696)

TABLE IX
NOISY $f^{(1)}$.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*						
$\sqrt{s_1}$	*		*	*			*		
s_2									
$\sqrt{s_2}$			*						
s_3	*	*	*	*			*	*	*
$\sqrt{s_3}$	*	*	*	*	*		*	*	*
s_c			*						
s_f	*	*	*	*			*		*
s_s	*	*	*	*				*	

TABLE X
NOISY $f^{(2)}$.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*						
$\sqrt{s_1}$	*		*	*				*	*
s_2									
$\sqrt{s_2}$			*						*
s_3	*	*	*	*			*	*	*
$\sqrt{s_3}$	*	*	*	*	*		*	*	*
s_c	*		*	*				*	*
s_f			*						
s_s			*						

The following observations can be made:

- Poor results are obtained for the harmonic function $f^{(3)}$ using all the objective functions tested. This is, however, in line with the results by Donoho and Johnstone [70]. They studied the least-square approximation errors of ridge approximation and kernel approximation, which are approximation techniques analogous to feedforward neural networks with a single hidden layer of sigmoidal units and a single hidden layer of radial basis function units, respectively. Focusing on the two-dimensional input space with respect to the Gaussian measure, they showed that ridge approximation is better for radial functions while kernel approximation is better for harmonic functions. Hence, in this case, networks of size larger than 15 may be needed for successive approximation and results for this problem are not conclusive.
- The testing performance of S_2 is poor for all the problems. But it can be significantly improved by using the modified version $\sqrt{S_2}$. Similar improvement also holds for S_1 and S_3. This shows that transforming the objective functions as discussed in Section IV-B.1 is beneficial.
- For the noiseless data, $\sqrt{S_3}$ has the smallest testing

TABLE XI
NOISY $f^{(3)}$.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*					*	*
$\sqrt{s_1}$			*					*	*
s_2									*
$\sqrt{s_2}$			*					*	*
s_3	*	*	*	*		*	*	*	*
$\sqrt{s_3}$			*					*	*
s_c			*					*	*
s_f			*						*
s_s									

TABLE XII
NOISY $f^{(4)}$.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*					*	*
$\sqrt{s_1}$			*					*	*
s_2									
$\sqrt{s_2}$	*	*	*				*	*	*
s_3	*	*	*					*	*
$\sqrt{s_3}$			*				*	*	*
s_c			*					*	*
s_f			*						*
s_s			*						

TABLE XIII
NOISY $f^{(5)}$.

	s_1	$\sqrt{s_1}$	s_2	$\sqrt{s_2}$	s_3	$\sqrt{s_3}$	s_c	s_f	s_s
s_1			*					*	*
$\sqrt{s_1}$	*		*	*	*	*	*	*	*
s_2									
$\sqrt{s_2}$			*					*	*
s_3			*					*	*
$\sqrt{s_3}$			*					*	*
s_c			*					*	*
s_f			*						*
s_s			*						

FVUs for the simple interaction function $f^{(1)}$ and the radial function $f^{(2)}$, and the second smallest for the additive function $f^{(4)}$. Its superiority over other objective functions is also confirmed by the pairwise comparison (Tables IV, V and VII). However, it is a bit inferior for the complicated interaction function $f^{(5)}$. From Figure 8e in Appendix C, one can see a number of outliers with high testing FVUs for $\sqrt{S_3}$. Moreover, the spreads of the testing FVUs for $\sqrt{S_3}$ are also comparatively larger than the other objective functions in most datasets. This probably indicates that superiority of $\sqrt{S_3}$ is marred by its sensitivity to the particular set of training data used.

- The testing performance of S_1 is much better than S_2, and $\sqrt{S_1}$ is generally better than $\sqrt{S_2}$. Except for the complicated interaction function $f^{(5)}$, $\sqrt{S_3}$ is also better than S_{cascor}. Hence, the claim in Section II on the difference in convergence rate is consistent with our experimental results.
- The superiority of S_3 over S_1, and $\sqrt{S_3}$ over $\sqrt{S_1}$ also supports the claim in Section II that ignoring the bias to the output unit during input training is beneficial.
- S_{fujita} has competitive performance only for the simple interaction function, and is significantly worse than most other objective functions for all other functions. This is probably because of the complicated form of S_{fujita}, making it hard to be optimized. The performance of S_{sqr} is also poor, as is expected following the discussion in Section II, suggesting that optimization based on S_{sqr} has to be repeated several times for satisfactory results, as has been done in [40].
- $\sqrt{S_1}$ and S_{cascor} have comparable testing performance. Moreover, the spreads of the testing FVUs are also comparable.
- For the noisy data, differences in performance between different objective functions become smaller. But, still $\sqrt{S_1}$, S_3 and $\sqrt{S_3}$ stand out among the others, as is evident from the pairwise comparison.

VI. Conclusion

In this paper, we study a number of objective functions for training new hidden units in constructive neural network learning algorithms. The aim is to derive a class of objective functions whose value and the corresponding weight updates can be computed in $O(N)$ time, where N is the number of training patterns. Moreover, even though input weight freezing is used in the constructive algorithm for computational efficiency, we require that the convergence property be preserved. This class of objective functions includes:

- $S_1 = (\sum_p E_p H_p)^2 / \sum_p H_p^2$,
- $S_2 = (\sum_p E_p H_p)^2$,
- $S_3 = (\sum_p (E_p - \bar{E})(H_p - \bar{H}))^2 / \sum_p (H_p - \bar{H})^2$,
- $\sqrt{S_1} = |\sum_p E_p H_p| / \sqrt{\sum_p H_p^2}$,
- $\sqrt{S_2} = |\sum_p E_p H_p|$,
- $\sqrt{S_3} = |\sum_p (E_p - \bar{E})(H_p - \bar{H})| / \sqrt{\sum_p (H_p - \bar{H})^2}$, and
- $S_{cascor} = |\sum_p (E_p - \bar{E})(H_p - \bar{H})|$.

The results here are not tied to a particular network architecture (the cascade-correlation architecture), and are useful to situations when complete re-training of the whole network is infeasible after hidden unit addition. Moreover, the constructive algorithm can easily be generalized to adding a group of hidden units, instead of just one, to the network simultaneously, by requiring this group to optimize these same objective functions.

Besides, we also propose a few computational tricks that can be used to improve the optimization of these objective functions. Specifically, we address the problem of slow convergence in plateaus of the objective function, which is caused by both the form of the objective function and the quickprop algorithm. A simple transformation of the objective function to be optimized, although theoretically identical to the original one, may lead to significantly improved results in the numerical optimization process. Handling of plateaus in quickprop is also modified, which is especially important for constructive neural network learning algorithms using the patience parameter.

In general, the generalization performance of networks using different objective functions can be influenced by a number of factors. Firstly, most constructive algorithms (such as those studied in this paper) use a greedy approach, by grabbing the largest possible residual error every time a new hidden unit is added. However, being the nature of any greedy approach, even if the "best" hidden unit function, as judged by the "best" objective function, is selected at each step, this does not necessarily guarantee that the final network will have the best performance.

Secondly, the objective functions used here are sample versions of their theoretical counterparts, basing on information available from the training set. Their accuracy compared to the true population version is thus dependent on whether the training set has sufficiently sampled the whole input space, and whether the noise in the training data has severely corrupted its true value, etc. In the latter case when large errors are to be expected, it may be reasonable to use robust versions of the objective functions [71]. Also, in some constructive algorithms (such as the cascade-correlation architecture [39]), the number of input weights to the new hidden unit keeps on increasing. In this case, regularization methods that add a penalty term to the original objective function may be beneficial.

Thirdly, practical optimization algorithms are prone to the local optimum problem. This plagues all objective functions being investigated here, but some are more vulnerable than the others, probably because of their complex functional form. Algorithms for finding global optima, such as simulated annealing, may of course be used, but the demand on computational resources will be much higher.

APPENDIX

A. Proofs

Proposition 1: For a fixed $g \in \Gamma$ ($\|g\| \neq 0$), the expression $\|f - (f_{n-1} + \beta g)\|$ achieves its minimum iff

$$\beta = \beta^* = \frac{\langle e_{n-1}, g\rangle}{\|g\|^2}.$$

Moreover, with β_n^* and β^* as defined above, $\|f - (f_{n-1} + \beta_n^* g_n)\| \leq \|f - (f_{n-1} + \beta^* g)\| \ \ \forall g \in \Gamma$, iff

$$\frac{\langle e_{n-1}, g_n\rangle^2}{\|g_n\|^2} \geq \frac{\langle e_{n-1}, g\rangle^2}{\|g\|^2} \ \ \forall g \in \Gamma.$$

Proof. Consider the case when all elements in Γ are of unit norm,

$$\begin{aligned} \Delta(g) &\equiv \|f - f_{n-1}\|^2 - \|f - (f_{n-1} + \beta g)\|^2 \\ &= \langle e_{n-1}, g\rangle^2 - (\langle e_{n-1}, g\rangle - \beta)^2. \end{aligned}$$

For a fixed g, the stated expression is minimized iff

$$\beta = \beta^* = \langle e_{n-1}, g\rangle,$$

with

$$\Delta_{max}(g) = \langle e_{n-1}, g\rangle^2.$$

From the set Γ, the stated expression is minimized when $\Delta_{max}(g)$ is maximized over all $g \in \Gamma$. Result follows when the assumption of unit norm is dropped. □

Theorem 1: Given span(Γ) is dense in L^2 and $\forall g \in \Gamma, 0 < \|g\| < b$ for some $b \in \Re$. If g_n is selected as to maximize $\langle e_{n-1}, g\rangle^2/\|g\|^2$, then $\lim_{n\to\infty} \|f - f_n\| = 0$.

Proof. Let $\hat{f}_n$ denote the version of f_n before output training.

$$\begin{aligned} \|e_{n-1}\|^2 - \|e_n\|^2 &= \|f - f_{n-1}\|^2 - \|f - f_n\|^2 \\ &\geq \|f - f_{n-1}\|^2 - \|f - \hat{f}_n\|^2 \\ &= \langle e_{n-1}, g_n\rangle^2/\|g_n\|^2, \end{aligned} \tag{20}$$

which is greater than zero as span(Γ) is dense (Lemma 3.3-7 of [72]). So, $\{\|e_n\|^2\}$ is strictly decreasing, and bounded below by zero, thus it converges. That is, $\forall \epsilon > 0, \exists N > 0$ such that when $m > n > N$,

$$\|e_n\|^2 - \|e_m\|^2 = \left|\|e_n\|^2 - \|e_m\|^2\right| < \epsilon.$$

As e_m is orthogonal to span$\{g_1, g_2, \ldots, g_m\}$ (Figure 7), therefore

$$\|e_n - e_m\|^2 = \|e_n\|^2 - \|e_m\|^2 < \epsilon,$$

i.e., $\{e_n\}$ is a Cauchy sequence. Because L^2 is complete, $\exists e \in L^2$ such that $e_n \to e$. From the fact that g_n maximizes (20) and $0 < \|g\| < b \ \ \forall g \in \Gamma$, therefore

$$\lim_{n\to\infty} \langle e_n, g\rangle^2/\|g\|^2 = 0 \Rightarrow \lim_{n\to\infty} \langle e_n, g\rangle = 0, \ \ \forall g \in \Gamma.$$

As strong convergence implies weak convergence, hence $\langle e, g\rangle = 0 \ \ \forall g \in \Gamma$, which implies $\|e\| = 0$. □

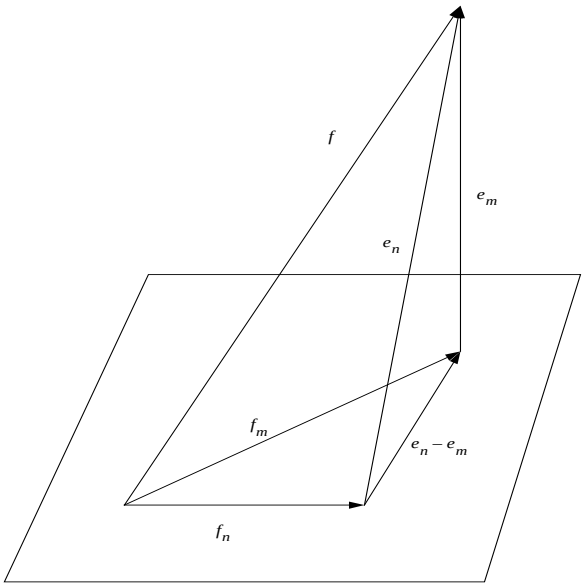


Fig. 7. Illustration for Theorem 1.

Proposition 2: If the target function is an exact summation of basis functions from Γ as in (5), and the network is constructed as in Theorem 1, then

$$\|e_n\|^2 < \frac{b_f^2 \|e_0\|^2}{n\|e_0\|^2 + b_f^2},$$

where $b_f \equiv m \max_{j=1,\ldots,m} |v_j| \|g_j^*\|$.

Proof. The proof follows that of [58]. Assume that the target function $f = \sum_{j=1}^m v_j g_j^*$ where $g_j^* \in \Gamma$. Consider

$$\begin{aligned} J &= \langle f, e_{n-1}\rangle \\ &= \langle e_{n-1} + f_{n-1}, e_{n-1}\rangle \\ &= \|e_{n-1}\|^2, \end{aligned}$$

as f_{n-1} is orthogonal to e_{n-1}. But J is also equal to

$$\sum_{j=1}^m v_j \langle g_j^*, e_{n-1}\rangle. \tag{21}$$

Hence, at least one of the terms in the summation of (21) must be greater than or equal to $\|e_{n-1}\|^2/m$, i.e.

$$\begin{aligned} & \max_{j=1,\ldots,m} v_j \langle g_j^*, e_{n-1}\rangle && \geq \frac{\|e_{n-1}\|^2}{m} \\ \Leftrightarrow \ & \max_{j=1,\ldots,m} v_j \|g_j^*\| \frac{\langle g_j^*, e_{n-1}\rangle}{\|g_j^*\|} && \geq \frac{\|e_{n-1}\|^2}{m} \\ \Rightarrow \ & \max_{j=1,\ldots,m} \frac{|\langle g_j^*, e_{n-1}\rangle|}{\|g_j^*\|} && \geq \frac{\|e_{n-1}\|^2}{m \max_{j=1,\ldots,m} |v_j| \|g_j^*\|} \\ & && = \frac{\|e_{n-1}\|^2}{b_f}. \end{aligned}$$

Hence, from (20) in Theorem 1, we have

$$\|e_{n-1}\|^2 - \|e_n\|^2 \geq \frac{\|e_{n-1}\|^4}{b_f^2}. \tag{22}$$

Substituting $\|e_n\|^2 = \frac{b_f^2}{z_n+1}$, (22) becomes

$$z_n \geq z_{n-1} + 1 + \frac{1}{z_{n-1}}.$$

By induction, we have

$$z_n \geq n + z_0 + \sum_{j=0}^{n-1} \frac{1}{z_j} > n + z_0.$$

Transforming back to $\|e_n\|^2$, we get

$$\|e_n\|^2 < \frac{b_f^2 \|e_0\|^2}{n\|e_0\|^2 + b_f^2}.$$

□

Proposition 3: If the target function is of the form in (5), $0 < \|g\| < b \ \ \forall g \in \Gamma$, and the network is constructed as in Corollary 1, then

$$\|e_n\|^2 < \frac{\bar{b}_f^2 \|e_0\|^2}{n\|e_0\|^2 + \bar{b}_f^2},$$

where $\bar{b}_f \equiv bm \max_{j=1,\ldots,m} |v_j|$

Proof. The proof is basically the same as that for Proposition 2, except now we have

$$\max_{j=1,\ldots,m} v_j \langle g_j^*, e_{n-1}\rangle \geq \frac{\|e_{n-1}\|^2}{m}$$

$$\Rightarrow \quad \max_{j=1,\ldots,m} |\langle g_j^*, e_{n-1}\rangle| \geq \frac{\|e_{n-1}\|^2}{m \max_{j=1,\ldots,m} |v_j|},$$

and

$$\begin{aligned} \|e_{n-1}\|^2 - \|e_n\|^2 &\geq \frac{\langle e_{n-1}, g_n\rangle^2}{\|g_n\|^2} \geq \frac{\langle e_{n-1}, g_n\rangle^2}{b^2} \\ &\geq \frac{\|e_{n-1}\|^4}{(bm \max_{j=1,\ldots,m} |v_j|)^2} \\ &= \frac{\|e_{n-1}\|^4}{\bar{b}_f^2}. \end{aligned}$$

□

Lemma 1: $\mathcal{H}/J$ is complete.

Proof. Suppose $\{\phi(g_n)\}$ is a Cauchy sequence in $\mathcal{H}/J$. There is a subsequence for which

$$\|\phi(g_{n_i}) - \phi(g_{n_{i+1}})\| < 2^{-i}, \ \ i = 1, 2, 3, \ldots, n_i < n_{i+1},$$

which implies $\|v_i - v_{i+1}\| = \|(v_i - \bar{v_i}) - (v_{i+1} - \bar{v_{i+1}})\| < 2^{-i}$ for some $v_i \in \phi(g_{n_i})$. Thus $\{v_i - \bar{v_i}\}$ is a Cauchy sequence in $\mathcal{H}$. Since $\mathcal{H}$ is complete, $\exists v \in \mathcal{H}$ s.t. $\|(v_i - \bar{v_i}) - v\| \to 0$, with $\bar{v} = \lim_{i\to\infty} \overline{v_i - \bar{v_i}} = 0$. Hence, $\|v_i - v\| \to 0$. It follows that $\phi(g_{n_i})$ converges to $\phi(v)$ in $\mathcal{H}/J$. But if a Cauchy sequence has a convergent subsequence, then the full sequence converges. Thus $\mathcal{H}/J$ is complete. □

Lemma 2: If span(Γ) is dense in $\mathcal{H}$, then span(Γ/J) is dense in $\mathcal{H}/J$.

Proof. For given $\phi(u) \in \mathcal{H}/J$ and $\epsilon > 0$, if span(Γ) is dense in $\mathcal{H}$, $\exists v \in span(\Gamma)$ s.t.

$$\|\phi(v) - \phi(g)\| = \|v - u\| < \epsilon,$$

by choosing u, v s.t. $\bar{u} = \bar{v} = 0$. Now, $\phi(v) \in span(\Gamma)/J = span(\Gamma/J)$. Hence, span($\Gamma/J$) is dense in $\mathcal{H}/J$. □

B. Nonlinear Output Units

In this section, we consider the case when the output unit transfer function is nonlinear. Nonlinear output units are commonly used in pattern classification tasks to restrict output values to ranges such as $[0, 1]$ or $[-1, 1]$. In the following, we denote the function implemented by the network as $\tau(f_n)$, where τ is Fréchet differentiable[11]. The corresponding residual error function e_n becomes $f - \tau(f_n)$.

Proposition 4: For a fixed $g \in \Gamma$ with $\|\tau'(f_{n-1})g\| \neq 0$, the expression $\|f - \tau(f_{n-1} + \beta g)\|$ achieves its minimum iff

$$\beta = \beta^* = \frac{\langle f - \tau(f_{n-1}), \tau'(f_{n-1})g\rangle}{\|\tau'(f_{n-1})g\|^2}, \tag{23}$$

under first-order approximation using Taylor series expansion. Moreover, with β^* as defined in (23),

$$\|f - \tau(f_{n-1} + \beta_n^* g_n)\| \leq \|f - \tau(f_{n-1} + \beta g)\|, \ \ \forall g \in \Gamma,$$

iff

$$\frac{\langle e_{n-1}, \tau'(f_{n-1})g_n\rangle^2}{\|\tau'(f_{n-1})g_n\|^2} \geq \frac{\langle e_{n-1}, \tau'(f_{n-1})g\rangle^2}{\|\tau'(f_{n-1})g\|^2} \ \ \forall g \in \Gamma.$$

Proof. Consider the case when all $g \in \Gamma$ satisfy the condition

$$\|\tau'(f_{n-1})g\| = 1. \tag{24}$$

$$\begin{aligned} \Delta(g) &\equiv \|f - \tau(f_{n-1})\|^2 - \|f - \tau(f_{n-1} + \beta g)\|^2 \\ &= 2\langle f, \tau(f_{n-1} + \beta g) - \tau(f_{n-1})\rangle - \|\tau(f_{n-1} + \beta g)\|^2 \\ &\quad + \|\tau(f_{n-1})\|^2. \end{aligned}$$

Ignoring higher-order terms in Taylor series expansion, $\tau(f_{n-1} + \beta g) \simeq \tau(f_{n-1}) + \tau'(f_{n-1})\beta g$. By linearity of τ' and (24), therefore,

$$\begin{aligned} \Delta(g) &\simeq \langle f - \tau(f_{n-1}), \tau'(f_{n-1})g\rangle^2 \\ &\quad - (\langle f - \tau(f_{n-1}), \tau'(f_{n-1})g\rangle - \beta)^2 \\ &= \langle e_{n-1}, \tau'(f_{n-1})g\rangle^2 - (\langle e_{n-1}, \tau'(f_{n-1})g\rangle - \beta)^2. \end{aligned}$$

Hence, for a fixed g, the stated expression is minimized (subject to the approximation in truncating the full Taylor series) iff

$$\beta = \beta^* = \langle e_{n-1}, \tau'(f_{n-1})g\rangle,$$

with

$$\Delta_{max}(g) = \langle e_{n-1}, \tau'(f_{n-1})g\rangle^2.$$

From the set Γ, the stated expression is minimized when $\Delta_{max}(g)$ is maximized over all $g \in \Gamma$. Result follows by removing the restriction (24). □

[11] Given X and Y are normed spaces. Suppose $f : X \to Y$ is defined on a neighborhood of $a \in X$. We say $f(h) = o(h)$ if $\|f(h)\|/\|h\| \to 0$ as $h \to 0$. A continuous linear operator $L : X \to Y$ is said to be the *Fréchet derivative* [73] of $f : X \to Y$ at the point $x \in X$ if

$$f(x + h) = f(x) + Lh + o(h),$$

as $h \to 0$. We write $L = f'(x)$.

Next, we compute the Fréchet derivative τ'. Let ψ be the output unit transfer function. Then $\tau(f_n)x = \psi(f_n(x))$, and

$$\begin{aligned}\tau(f_n + h)x - \tau(f_n)x &= \psi((f_n + h)(x)) - \psi(f_n(x)) \\ &= \psi(f_n(x) + h(x)) - \psi(f_n(x)) \\ &= h(x)\psi'(f_n(x)) + o(h(x)).\end{aligned}$$

Thus, $(\tau'(f_n)h)(x) = h(x)\psi'(f_n(x))$, and the corresponding objective function to be maximized for nonlinear output units is:

$$S_4 = \frac{(\sum_p E_p \psi'_p H_p)^2}{\sum_p \psi'^2_p H_p^2},$$

where ψ'_p is the derivative of the output unit transfer function for pattern p. In the special case when the output unit transfer function is linear, $\tau(f_n) = f_n$, Proposition 4 reduces to Proposition 1, and S_4 reduces to S_1. However, unlike the case for linear output, convergence proof cannot be derived because approximation by Taylor series expansion is used.

C. Boxplots of the Simulation Results

In this section, we report the boxplots of the generalization performance of networks constructed using different objective functions. The experimental setting has been described in Section V-A. In each boxplot, the horizontal line in the interior of the box is located at the median. The height of the box is equal to the interquartile distance, or IQD, which is the difference between the third quartile of the data and the first quartile. The *whiskers* (the dotted lines extending from the top and bottom of the box) extend to the extreme values or a distance $1.5 \times IQD$ from the center, whichever is less. Data points which fall outside the whiskers are indicated by horizontal lines.

Acknowledgments

This research is partially supported by the Hong Kong Telecom Institute of Information Technology under grant HKTIIT 92/93.002 awarded to the second author.

References

[1] R. Reed, "Pruning algorithms – a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, Sept. 1993.

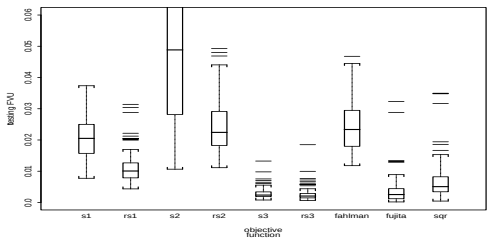
[2] E. Fiesler, "Comparative bibliography of ontogenic neural networks," in *Proceedings of the International Conference on Artificial Neural Networks*, Sorrento, Italy, May 1994, vol. 1, pp. 793–796.

[3] T.Y. Kwok and D.Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," 1996, To appear in *IEEE Transactions on Neural Networks*.

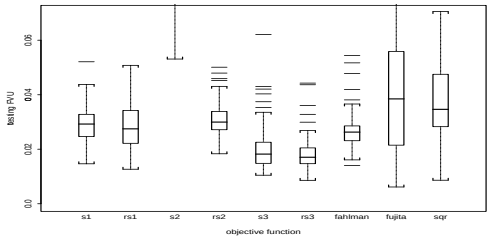
[4] D.E. Nelson and S.K. Rogers, "A taxonomy of neural network optimality," in *Proceedings of the IEEE National Aerospace and Electronics Conference*, Dayton, OH, USA, May 1992, vol. 3, pp. 894–899.

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

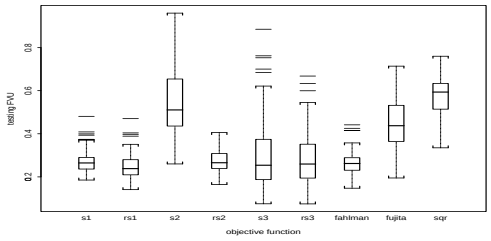
[6] S.J. Hanson and L.Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Advances in Neural Information Processing Systems 1*, D.S. Touretzky, Ed., pp. 177–185. Morgan Kaufmann, San Mateo, CA, 1989.



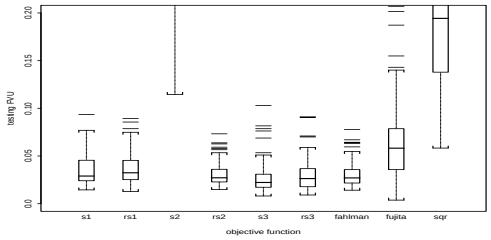
(a) $f^{(1)}$



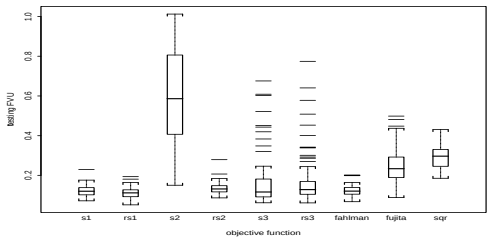
(b) $f^{(2)}$



(c) $f^{(3)}$

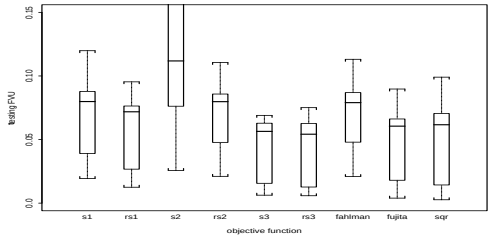


(d) $f^{(4)}$

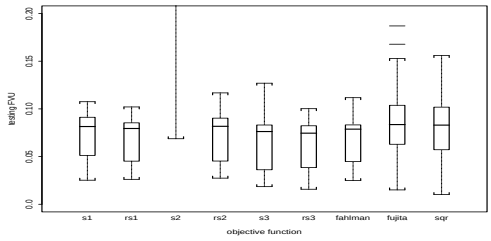


(e) $f^{(5)}$

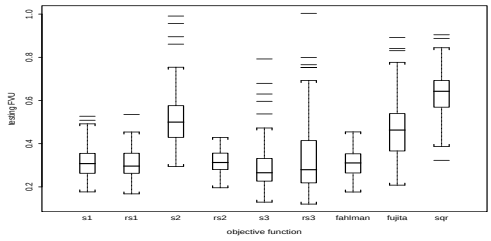
Fig. 8. Comparison of the logarithm of the testing FVUs on noiseless data. Here, s1 stands for S_1, rs1 for $\sqrt{S_1}$, s2 for S_2, rs2 for $\sqrt{S_2}$, s3 for S_3, rs3 for $\sqrt{S_3}$, fahlman for S_{cascor}, fujita for S_{fujita}, and sqr for S_{sqr}.



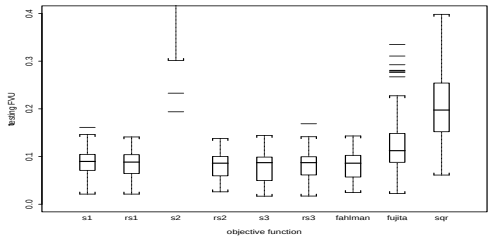
(a) $f^{(1)}$



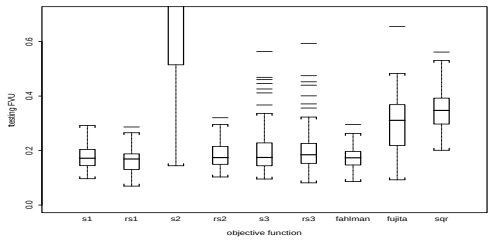
(b) $f^{(2)}$



(c) $f^{(3)}$



(d) $f^{(4)}$



(e) $f^{(5)}$

Fig. 9. Comparison of the logarithm of the testing FVUs on noisy data.

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

[8] G.H. Golub, M. Heath, and G. Wahba, "Generalized cross-validation as a method for choosing a good ridge parameter," *Technometrics*, vol. 21, no. 2, pp. 215–223, 1979.

[9] W.L. Buntine and A.S. Weigend, "Bayesian back-propagation," *Complex Systems*, vol. 5, pp. 603–643, 1991.

[10] D.J.C. MacKay, "Bayesian interpolation," *Neural Computation*, vol. 4, no. 3, pp. 415–447, May 1992.

[11] R.M. Neal, *Bayesian Learning for Neural Networks*, Ph.D. thesis, Department of Computer Science, University of Toronto, 1995.

[12] H. H. Thodberg, "A review of Bayesian neural networks with an application to near infrared spectroscopy," *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 56–72, 1996.

[13] P.M. Williams, "Bayesian regularization and pruning using a Laplace prior," *Neural Computation*, vol. 7, pp. 117–143, 1995.

[14] D.J.C. MacKay, "A practical Bayesian framework for back-propagation networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, May 1992.

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

[16] G. Deffuant, "Neural units recruitment algorithm for generation of decision trees," in *Proceedings of the 1990 IEEE International Joint Conference on Neural Networks*, San Diego, CA, USA, June 1990, vol. 1, pp. 637–642.

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

[18] M. Marchand, M. Golea, and P. Ruján, "A convergence theorem for sequential learning in two-layer perceptrons," *Europhysics Letters*, vol. 11, no. 6, pp. 487–492, 1990.

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

[20] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. AC-19, no. 6, pp. 716–723, Dec. 1974.

[21] A. Barron, "Predicted squared error: A criterion for automatic model selection," in *Self-Organizing Methods in Modeling*, S. Farlow, Ed. Marcel Dekker, New York, 1984.

[22] P. Craven and G. Wahba, "Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation," *Numerische Mathematik*, vol. 31, pp. 377–403, 1979.

[23] J.E. Moody, "Note on generalization, regularization, and architecture selection in nonlinear learning systems," in *Neural Networks for Signal Processing. Processing of the 1991 IEEE Workshop*, B.H. Juang, S.Y. Kung, and C.A. Kamm, Eds., Princeton, NJ, USA, Sept. 1991, pp. 1–10.

[24] J. Rissanen, "Modelling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1975.

[25] G. Schwartz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, pp. 461–464, 1978.

[26] J. Moody, "Prediction risk and architecture selection for neural networks," in *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, V. Cherkassky, J.H. Friedman, and H. Wechsler, Eds., vol. 136 of *NATO ASI Series F*, pp. 147–165. Springer-Verlag, 1994.

[27] M. Stone, "Cross-validatory choice and assessment of statistical predictions (with discussion)," *Journal of the Royal Statistical Society Series B*, vol. 36, pp. 111–147, 1974.

[28] A.S. Weigend and B. LeBaron, "Evaluating neural network predictors by bootstrapping," in *Proceedings of International Conference on Neural Information Processing*, Seoul, Korea, Oct. 1994, vol. 2, pp. 1207–1212.

[29] B.D. Ripley, "Choosing network complexity," in *Probabilistic Reasoning and Bayesian Belief Networks*, A. Gammerman, Ed., pp. 97–108. Alfred Waller, 1995.

[30] B.D. Ripley, "Statistical ideas for selecting network architectures," in *Neural Networks: Artificial Intelligence and Indus-*

trial Applications, B. Kappen and S. Gielen, Eds., pp. 183–190. Springer, 1995.

[31] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Science*, vol. 1, no. 4, pp. 365–375, 1989.

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

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

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

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

[36] K. Hornik, "Some new results on neural network approximation," *Neural Networks*, vol. 6, pp. 1069–1072, 1993.

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

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

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

[40] 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*, vol. 5, no. 3, pp. 342–353, May 1994.

[41] E. Littmann and H. Ritter, "Cascade LLM networks," in *Proceedings of the International Conference on Artificial Neural Networks*, Brighton, UK, Sept. 1992, vol. 1, pp. 253–257.

[42] A. Saha, C.L. Wu, and D.S. Tang, "Approximation, dimension reduction, and nonconvex optimization using linear superpositions of Gaussians," *IEEE Transactions on Computers*, vol. 42, no. 10, pp. 1222–1233, Oct. 1993.

[43] Y. Zhao and C.G. Atkeson, "Some approximation properties of projection pursuit learning networks," in *Advances in Neural Information Processing Systems 4*, J.E. Moody, S.J. Hanson, and R.P. Lippmann, Eds., pp. 936–943. Morgan Kaufmann, San Mateo, CA, 1992.

[44] J.L. Yuan and T.L. Fine, "Forecasting demand for electric power," in *Advances in Neural Information Processing Systems 5*, S.J. Hanson, J.D. Cowan, and C.L. Giles, Eds., pp. 739–746. Morgan Kaufmann, San Mateo, CA, 1993.

[45] J.H. Friedman, "Exploratory projection pursuit," *Journal of the American Statistical Association*, vol. 82, no. 397, pp. 249–266, Mar. 1987.

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

[47] E.B. Baum and D. Haussler, "What size net gives valid generalization?," in *Advances in Neural Information Processing Systems 1*, D.S. Touretzky, Ed., pp. 81–90. Morgan Kaufmann, San Mateo, CA, 1989.

[48] P. Courrieu, "A convergent generator of neural networks," *Neural Networks*, vol. 6, no. 6, pp. 835–844, 1993.

[49] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.

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

[51] 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*, vol. 3, no. 5, pp. 672–682, Sept. 1992.

[52] 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*, Baltimore, MD, USA, June 1992, vol. 1, pp. 800–805.

[53] 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*, London, UK, Oct. 1992, pp. 5/1–4.

[54] 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*, Nimes, France, Nov. 1992, pp. 455–464.

[55] 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*, Hillsdale, NJ, 1991, pp. 635–640.

[56] D.G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, 2nd edition, 1989.

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

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

[59] A.R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, May 1993.

[60] L.K. Jones, "A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training," *The Annals of Statistics*, vol. 20, no. 1, pp. 608–613, 1992.

[61] L.K. Jones, "On a conjecture of Huber concerning the convergence of projection pursuit regression," *The Annals of Statistics*, vol. 15, no. 2, pp. 880–882, 1987.

[62] C. Darken, M. Donahue, L. Gurvits, and E. Sontag, "Rate of approximation results motivated by robust neural network learning," Tech. Rep., Siemens Corporate Research, Inc., Princeton, New Jersey, Apr. 1994.

[63] V. Kůrková and B. Beliczynski, "Incremental approximation by one-hidden-layer neural networks," in *Proceedings of the International Conference on Artificial Neural Networks*, Paris, France, Oct. 1995, vol. 1, pp. 505–510.

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

[65] 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.

[66] R.S. Crowder, "Cascor.c, C implementation of the cascade-correlation learning algorithm," 1990.

[67] R.A Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295–307, 1988.

[68] T.Y. Kwok and D.Y. Yeung, "Constructive neural networks: Some practical considerations," in *Proceedings of the IEEE International Conference on Neural Networks*, Orlando, Florida, USA, June 1994, vol. 1, pp. 198–203.

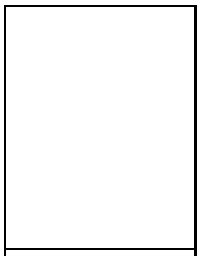
[69] V.K. Rohatgi, *Statistical Inference*, John Wiley & Sons, 1984.

[70] D.L. Donoho and I.M. Johnstone, "Projection-based approximation and a duality with kernel methods," *The Annals of Statistics*, vol. 17, no. 1, pp. 58–106, 1989.

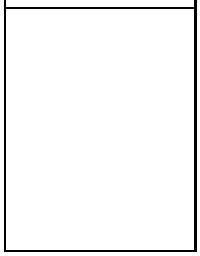
[71] P.J. Huber, *Robust Statistics*, John Wiley & Sons, 1981.

[72] E. Kreyszig, *Introductory Functional Analysis with Applications*, John Wiley & Sons, New York, Wiley Classics Library edition, 1989.

[73] D.H. Griffel, *Applied Functional Analysis*, Ellis Horwood, 1981.



Tin-Yau Kwok received his Ph.D. degree in computer science from the Hong Kong University of Science and Technology. His research interests include the theory and applications of artificial neural networks, pattern recognition, machine learning, and data mining.



Dit-Yan Yeung received his B.Sc.(Eng.) degree in electrical engineering and M.Phil. degree in computer science from the University of Hong Kong, and the Ph.D. degree in computer science from the University of Southern California. He is currently an assistant professor in the Hong Kong University of Science and Technology. His current research interests include neural computation, statistical learning theory, and handwriting recognition.