Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

NEUROCOMPUTING

Neurocomputing 51 (2003) 361–385

www.elsevier.com/locate/neucom

A new strategy for adaptively constructing multilayer feedforward neural networks ☆

L. Ma, K. Khorasani*

Department of Electrical and Computer Engineering, Concordia University, Montreal, Que., Canada H3G 1M8

Received 28 July 2001; accepted 9 March 2002

Abstract

In this paper a new strategy for adaptively and autonomously constructing a multi-hidden-layer feedforward neural network (FNN) is introduced. The proposed scheme belongs to a class of structure level adaptation algorithms that adds both new hidden units *and* new hidden layers one at a time when it is determined to be needed. Using this strategy, a FNN may be constructed having as many hidden layers and hidden units as required by the complexity of the problem being considered. Simulation results applied to regression problems are included to demonstrate the performance capabilities of the proposed scheme.

Keywords: Constructive networks; Multilayer feedforward neural networks; Regression and function approximation

1. Introduction

Among the existing numerous neural networks (NNs) paradigms, such as Hopfield networks, Kohonen's self-organizing feature maps (SOM), etc. the feedforward NNs (FNNs) are the most popular due to their flexibility in structure, good representational capabilities, and large number of available training algorithms [3,20,21,29]. In this paper we will be mainly concerned with FNNs.

☆ This research was supported in part by the NSERC (Natural Science and Engineering Research Council of Canada) research grant RGPIN-42515.

* Corresponding author. Tel.: +1-514-848-3086; fax: +1-514-848-2802.
E-mail address: kash@ece.concordia.ca (K. Khorasani).

0925-2312/03/$ - see front matter
PII: S0925-2312(02)00597-0

When using a NN, one needs to address three important issues. The solutions to these will significantly influence the overall performance of the NN as far as the following two considerations are concerned: (i) recognition rate to training patterns, and (ii) generalization performance to new patterns and data sets that have not been presented during network training.

The first problem is the selection of data/patterns for network training. This is a problem that has practical implications and has not received as much attention as necessary by neural networks researchers. The training data set selection can have considerable effects on the performance of the trained network. Some research on this issue has been conducted in [33] (and the references therein).

The second problem is the selection of an appropriate and efficient training algorithm from a large number of possible training algorithms that have been developed in the literature, such as the classical error backpropagation (BP) [27] and its many variants [29,22,32] and the second-order algorithms [31,25], to name a few. Many new training algorithms with faster convergence properties and less computational requirements are being developed by researchers in the NN community.

The third problem is the determination of the network size. This problem is more important from a practical point of view when compared to the above two problems, and is generally more difficult to solve. The problem here is to find a network structure as small as possible to meet certain desired performance specifications. What is usually done in practice is that the developer trains a number of networks with different sizes, and then the smallest network that can fulfill all or most of the required performance requirements is selected. This amounts to a tedious process of trial and errors that seems to be unfortunately unavoidable. This paper focuses on developing a systematic procedure for an automatic determination and/or adaptation of the network architecture for a FNN.

The 2nd and 3rd problems are actually closely related to one another in the sense that different training algorithms are suitable for different NN topologies. Therefore, the above three considerations are indeed critical when a NN is to be applied to a real-life problem. Consider a data set generated by an underlying function. This situation usually occurs in pattern classification, function approximation, and regression problems. The problem is to find a model that can represent the input–output relationship of the data set. The model is to be determined or trained based on the data set so that it can predict within some prespecified error bounds the output to any new input pattern. In general, a FNN can solve this problem if its structure is chosen appropriately. Too small a network may not be able to learn the inherent complexities present in the data set, but too large a network may learn "unimportant" details such as observation noise in the training samples, leading to "overfitting" and hence poor generalization performance. This is analogous to the situation when one uses polynomial functions for curve fitting problems. Generally acceptable results cannot be achieved if too few coefficients are used, since the characteristics or features of the underlying function cannot be captured completely. However, too many coefficients may not only fit the underlying function but also the noise contained in the data, yielding a poor representation of the underlying function. When an "optimal" number of coefficients are used, the fitted polynomial will then yield the "best" representation of the function and also the "best" prediction for any new data.

A similar situation arises in the application of neural networks, where it is also imperative to relate the architecture of the NN to the complexity of the problem. Obviously, algorithms that can determine an appropriate network architecture automatically according to the complexity of the underlying function embedded in the data set are very cost-efficient, and thus highly desirable. Efforts toward the network size determination have been made in the literature for many years, and many techniques have been developed [12,17] (and the references therein]. Towards this end, in the next section, we review three general methods that deal with the problem of automatic NN structure determination.

2. Adaptive structure neural networks

2.1. Pruning algorithms

One intuitive way to determine the network size is to first establish by some means a network that is considered to be sufficiently large for the problem being considered, and then trim the unnecessary connections or units of the network to reduce it to an appropriate size. This is the basis for the pruning algorithms. Since it is much "easier" to determine or select a "very large" network than to find the proper size needed, the pruning idea is expected to provide a practical but a partial solution to the structure determination issue. The main problem to resolve is how to design a criterion for trimming or pruning the redundant connections or units in the network. Mozer and Smolensky [23] proposed a method that estimates which units are "least important" and deletes them during training. Karnin [14] defined a sensitivity of the error function being minimized with respect to the removal of each connection and pruned the connections that have low sensitivities. Le Cun et al. [8] designed a criterion called "saliency" by estimating the second derivative of the error function with respect to each weight, and trimmed the weights with sensitivities lower than some prespecified bound. Castellano et al. [5] developed a new iterative pruning algorithm for FNNs, which does not need the problem-dependent tuning phase and the retraining phase that are required by other pruning algorithms such as those in [8,23]. The above pruning algorithms share the following limitations: (i) the size of the initial network may be difficult to determine a priori, and (ii) the computational costs are very excessive due to the fact that repeated pruning and retraining processes have to be performed.

2.2. Regularization

A neural network that is larger than required will, in general, have some weights that have very little, if anything, to do with the essential representation of the input–output relationship. Although, they may contribute in reducing the training error, they will not behave properly when the network is fed with new input data, resulting in an increase of the output error in an unpredictable fashion. This is the case when the trained NN does not generalize well. In the conventional BP and its many variants these weights along with other more crucial weights are all adjusted in the training

phase. Pruning these unnecessary weights from the trained network according to a certain sensitivity criterion, as mentioned above, is one possible way to deal with this situation. Alternatively, by invoking a scheme known as regularization, one may impose certain conditions on the NN training so that the unnecessary weights will be automatically forced to converge to zero. This can be achieved by adding a penalty term to the error objective/cost function that is being minimized. Several penalty terms are proposed in the literature that are found to be effective for different problems, see for example the Refs. [7,13].

However, the regularization techniques still cannot automatically determine the size of the network. The initial network size has to be specified a priori by the user. Furthermore, in the objective/cost function one has to appropriately weight the error and the penalty terms. This is controlled by a parameter known as the regularization parameter. Usually, this parameter is selected by trial and error or by some heuristic based procedure. Recently, the Bayesian methods have been incorporated into network training for automatically selecting the regularization parameter [4,34]. In these methods, pre-assumed priors regarding the network weights such as normal or Laplace distributions are used that favor small network weights. However, the relationship between the generalization performance and the Bayesian evidence has not been established, and those priors sometimes do not produce good results.

2.3. Constructive learning algorithms

The third approach for determining the network size is known as constructive learning. Constructive learning alters the network structure as learning proceeds, producing automatically a network with an appropriate size. In this approach one starts with an initial network of a "small" size, and then incrementally add new hidden units and/or hidden layers until some prespecified error requirement is reached, or no performance improvement can be observed. The network obtained this way is a "reasonably" sized one for the given problem at hand. Generally, a "minimal" or an "optimal" network is seldom achieved by using this strategy, however a "sub-minimal/sub-optimal" network can be expected [17,18]. This problem has attracted a lot of attention by many researchers and several promising algorithms have been proposed in the literature. Kwok and Yeung in [17] surveys the major constructive algorithms in the literature. Dynamic node creation algorithm and its variants [2,30], activity-based structure level adaptation [19,35], Cascade-Correlation (CC) algorithms [10,26], and the constructive one-hidden-layer (OHL) algorithms [16,18] are among the most common constructive learning algorithms developed so far in the literature.

Constructive algorithms have the following major advantages over the pruning algorithms and regularization-based techniques:

*A*1. It is easier to specify the initial network architecture in constructive learning techniques, whereas in pruning algorithms one usually does not know a priori how large the original network should be.

*A*2. Constructive algorithms tend to build small networks due to their incremental learning nature. Networks are constructed that have direct correspondence to the com-

plexity of the given problems and the specified performance requirements, while excessive efforts may be rendered to trim the unnecessary weights in the network in pruning algorithms. Thus, constructive algorithms are generally more efficient (in terms of training time and network complexity/structure) than pruning algorithms.

*A*3. In pruning algorithms and regularization-based techniques, one must specify or select several problem-dependent parameters in order to obtain an "acceptable" and "good" network yielding satisfactory performance results. This aspect could potentially reduce the applicability of these algorithms in real-life applications. On the other hand, constructive algorithms do not share these limitations.

3. Constructive algorithms for feedforward neural networks

In this section, we first give a simple formulation of the training problem for a constructive one-hidden-layer FNN in the context of a nonlinear optimization problem. The advantages and disadvantages of these constructive algorithms are also discussed.

3.1. Formulation of constructive FNN training

Suppose a FNN is used to approximate a regression function whose input vector (or predictor variables) is indicated by the multi-dimensional vector $\mathbf{X}$ and its output (or response) is expressed by the scalar Y, without loss of any generality. A regression surface (input–output function) $g(\cdot)$ is used to describe the relationship between $\mathbf{X}$ and Y. A FNN is trained and used to realize or represent this relationship. The input samples are denoted by $(\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^P)$, the output samples are denoted by $(y_1^j, \ldots, y_{L-1}^j, y_L^j)$, $j = 1, \ldots, P$, and the corresponding target samples (or observations) are denoted by $(d^1, d^2, \ldots, d^P)$, which are the output data contaminated by an additive white noise vector $\mathbf{\Lambda} = (\varepsilon^1, \varepsilon^2, \ldots, \varepsilon^P)$, where $L-1$ is the number of hidden layers, and L denotes the output layer, and P is the number of patterns in the data set. The network training may be formulated as the following unconstrained least square (LS) nonlinear optimization problem

$$\min_{L,\mathbf{n},\mathbf{f},\mathbf{w}} \sum_{j=1}^{P} (d^j - y_L^j)^2 \tag{1}$$

subject to

$$\begin{aligned}
&\mathbf{y}_1^j = f_1(\mathbf{w}_1 \mathbf{x}^j), \quad \mathbf{w}_1 \in \Re^{n_1 \times M},\ \mathbf{y}_1^j \in \Re^{n_1},\ \mathbf{x}^j \in \Re^M,\\
&\mathbf{y}_2^j = f_2(\mathbf{w}_2 \mathbf{y}_1^j), \quad \mathbf{w}_2 \in \Re^{n_2 \times n_1},\ \mathbf{y}_2^j \in \Re^{n_2},\\
&\vdots\\
&y_L^j = f_L(\mathbf{w}_L \mathbf{y}_{L-1}^j), \quad \mathbf{w}_L \in \Re^{1 \times n_{L-1}},\ y_L^j \in \Re^1,
\end{aligned} \tag{2}$$

where $\mathbf{n}=(n_1, n_2, \ldots, n_{L-1})$ is a vector denoting the number of hidden units at each hidden layer, $\mathbf{f}=(f_1, f_2, \ldots, f_L)$ denotes the activation function of each layer, $f_1, f_2, \ldots, f_{L-1}$ are usually nonlinear activation functions, with f_L the activation function of the output layer selected to be linear for a regression problem, and $\mathbf{w}=(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_L)$ represents the weight matrix corresponding to each layer. All levels of adaptation, that is structure level, functional level, and learning level are included in the above LS nonlinear optimization problem. The performance index (1) is clearly too complicated to solve by existing optimization techniques. Fortunately, by fixing certain variables, the optimization problem will become easier to solve. However, only a suboptimal solution can then be provided. As it turns out, a suboptimal solution suffices in many practical situations.

Practically, one can solve (1)–(2) only through an incremental procedure. That is, by first fixing certain variables, say the activation functions and the number of hidden layers and units, and then by solving the resulting LS optimization problem with respect to the remaining variables. This process is to be repeated until an acceptable solution or a network is obtained. In this way, for a given choice of activation functions and the number of units, (1)–(2) reduces to a nonlinear optimization problem with respect to the number of hidden layers and weight matrices. The problem of weight selection may be solved by using the steepest-descent (for example, the BP) type or second-order (Quasi-Newton) recursive methods. The error surface represented by (1)–(2) has, in general, many local minima and the solution to the problem is sought for only one local minimum at a time. The error surface changes its shape each time the activation functions and/or the number of hidden layers and/or units are modified, resulting in different local minima each time.

The standard constructive algorithms available in the literature have adopted the above approach and can therefore provide only suboptimal solutions to (1)–(2). The constructive multilayer FNN proposed in this paper provides also a suboptimal solution to (1)–(2).

3.2. Limitations of the current constructive FNNs

Our motivation for developing the proposed constructive learning algorithm may be stated due to the following reasons:

*R*1. The FNN is simple and elegant in structure. The fan-in problem with the Cascade-Correlation (CC)-type architectures is not present in this structure. Furthermore, as "deeper" the CC structure becomes, the more input-side connections for a new hidden unit will be required. This may give rise to degradation of generalization performance of the network, as some of the connections may become irrelevant to the prediction of the output.

*R*2. FNN is a universal approximator as long as the number of hidden units is sufficiently large. Therefore, the convergence of constructive algorithms can be easily established [18].

*R*3. The constructive learning process is simple and facilitates the investigation of the training efficiency and development of other improved constructive strategies.

The constructive multi-hidden-layer FNN considered in this paper may yield improved approximation and representation capabilities to many complicated problems when compared to OHL-FNNs. Other multi-layer architectures have also been developed in the literature, such as the stack learning algorithm [11] and adding-and-deleting algorithm [24]. The stack learning algorithm begins with a minimal structure consisting of the input and the output units only, similar to the initial network in CC. The algorithm then constructs a network by creating a new set of output units and by converting the previous output units into new hidden units. The new output layer has connections to both the original input units and all the established hidden units. In other words, this algorithm generates a network that has a similar structure as in the CC-based networks, and hence has the same limitations as the CC. In the adding-and-deleting algorithm, the network training is divided into two phases: addition phase and deletion phase. These two phases are controlled by evaluating the network performance. The so-called backtracking technique is used to avoid the "limit off" of the constructive learning. This algorithm may produce multi-layered FNNs, but it is computationally very intensive due to its lengthy addition-and-deletion process and the use of the BP-based training algorithm.

4. Proposed multi-hidden-layer FNN strategy

Construction and design of multi-hidden-layer networks are generally more difficult than those of one-hidden-layer (OHL) networks since one has to determine not only the depth of the network in terms of the number of hidden layers, but also the number of neurons in each layer. In this section, a new strategy for constructing a multi-hidden-layer FNN with regular connections is proposed by extending the strategy that was developed for constructive OHL-FNNs in [18]. The new constructive algorithm has the following features and advantages:

(1) As with other OHL constructive algorithms, the number of hidden units in a given hidden layer is automatically determined by the algorithm. In addition, the number of hidden layers are also determined automatically by the algorithm.
(2) The constructed multi-hidden-layer network has regular connections. The *fan-in* problem with the Cascade-Correlation (CC) and the stack learning algorithms has been eliminated. In other words, the new constructive network scales up appropriately.
(3) The algorithm adds hidden units and hidden layers one at a time, when and if they are needed. The input-side weights for all the hidden units in the installed hidden layer are fixed (input weights are frozen), however, the input-side weights for a new hidden unit that is being added to the network are updated during the input-side training. Generally, a constructive OHL may need to grow a large number of hidden units, or even could fail to represent a map that actually requires more than a single hidden layer. Therefore, it is expected that our proposed algorithm that incrementally expands its hidden layers and units automatically as a function of the complexity of the problem, would converge very fast

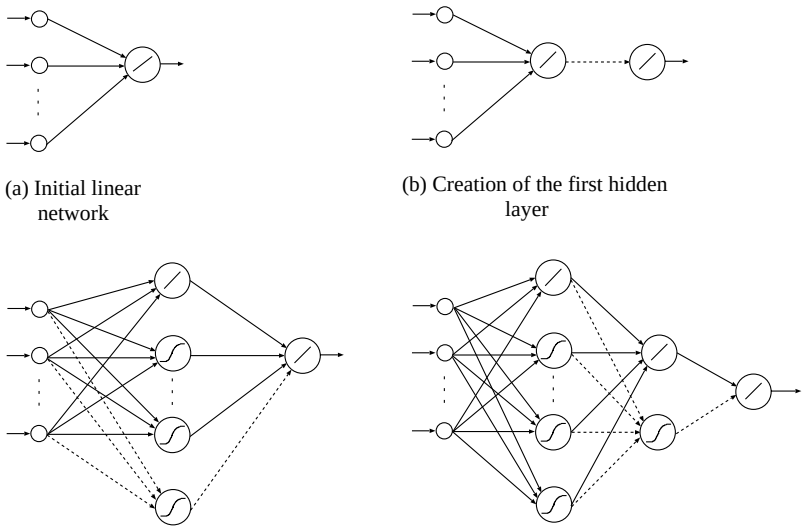


Fig. 1. Multi-hidden-layer network construction process for the proposed strategy. (a) Initial linear network; (b) Creation of the first hidden layer; (c) Addition of hidden units to the first hidden layer; (d) Creation of the second hidden layer and its units.

and is computationally more efficient when compared to other constructive OHL networks.

(4) Incentives are also introduced in the proposed algorithm to control and monitor the hidden layer creation process such that a network with fewer layers are constructed. This will encourage the algorithm to solve problems using smaller networks, reducing the costs of network training and implementation.

4.1. Construction, design and training of the proposed algorithm

For sake of simplicity, and without loss of any generality, suppose one desires to construct a FNN to solve a mapping or a regression problem from a multi-dimensional input space to a single dimensional output space. Extension to the case of multi-input multi-output spaces is quite trivial and straightforward. It is also assumed that the output unit has a linear activation function for dealing with the regression problem, although nonlinear activation function has to be used for pattern recognition problem. Specifically, the output-side training of the weights would have to be performed by utilizing the Delta rule, instead of an LS solution that is used here with the activation function selected as a linear function. The proposed constructive algorithm consists of the following steps (see Fig. 1):

Step 1: Initially train the network with a single output node and no hidden layers (refer to Fig. 1(a)). The training of the weights is achieved according to a LS solution

or some gradient-based algorithm, including the bias of the output node. The training error FVU (as described below) is monitored and if it is not smaller than some pre-specified required threshold, one would then proceed to Step 2a, otherwise one would go to Step 2c. The performance of the network is measured by the fraction of variance unexplained (FVU) [18] which is defined as

$$\mathrm{FVU} = \frac{\sum_{j=1}^{P} (\hat{g}(x^j) - g(x^j))^2}{\sum_{j=1}^{P} (g(x^j) - \bar{g})^2}, \tag{3}$$

where $g(\cdot)$ is the function to be implemented by the FNN, $\hat{g}(\cdot)$ is an estimate of $g(\cdot)$ realized by the network, and $\bar{g}$ is the mean value of $g(\cdot)$. The FVU is equivalently a ratio of the error variance to the variance of the function being analyzed by the network. Generally, the larger the variance of the function, the more difficult it would be to do the regression analysis. Therefore, the FVU may be viewed as a measure normalized by the "complexity" of the function. Note that the FVU is proportional to the mean square error (MSE). Furthermore, the function under study is likely to be contaminated by an additive noise ε. In this case the signal-to-noise ratio (SNR) is defined by

$$\mathrm{SNR} = 10 \log_{10} \left\{ \frac{\sum_{j=1}^{P} (g(x^j) - \bar{g})^2}{\sum_{j=1}^{P} (\varepsilon^j - \bar{\varepsilon})^2} \right\}, \tag{4}$$

where $\bar{\varepsilon}$ is the mean value of the additive noise, which is usually assumed to be zero.

Step 2a: Treat the output node as a hidden unit with unit output-side weight and zero bias (refer to Fig. 1(b)). The hidden unit input-side weights are fixed and are to be determined from the previous training step. The output error of the new linear output node will be identical to its hidden unit. Proceed to Step 2b.

Step 2b: A new hidden unit is added to the present layer in Step 2a one at a time, just as in a constructive OHL-FNN (refer to Fig. 1(c)). Each time a new hidden unit is added, the output error (FVU) is monitored to see if it is smaller than a certain prescribed threshold value. If so, one then proceeds to Step 2c. Otherwise, a new hidden unit is added until the output error FVU is stabilized above the prescribed threshold. Once the error is stabilized, Step 2a is invoked again and a new hidden layer is introduced to possibly further reduce the output error (refer to Fig. 1(d)). Steps 2a and 2b are repeated as many times as necessary until the output error FVU is below the prescribed desired threshold, following which one then proceeds to Step 2c. Input-side training of a new hidden unit weights is performed by maximizing a correlation-based objective function (details are provided in Appendix A), whereas the output-side training of the weights is performed through a LS type algorithm.

Step 2c: The constructive training is terminated.

For implementing the above algorithm some further specifications are needed. First, a metric "stop-ratio" is introduced to determine if the training error FVU is stabilized

as a function of the added hidden unit, namely

$$stop\text{-}ratio = \frac{\mathrm{FVU}(n-1) - \mathrm{FVU}(n)}{\mathrm{FVU}(n-1)} \times 100\%, \tag{5}$$

where FVU(n) denotes the training error FVU when the nth hidden unit is added to a hidden layer of the network. Note that according to [17] addition of a proper hidden unit to an existing network should reduce the training error. An issue that needs addressing here is the magnitude selection of the stop-ratio. At the early stages of the constructive training, this ratio may be large. However, as more hidden units are added to the network the ratio will monotonically decrease. When it has become "sufficiently" small, say a few percents, the inclusion of further additional hidden units will contribute little to the reduction of the training error, and as a matter of fact may actually start giving rise to an increase in the generalization error, as the unnecessary hidden units may force the network to begin to memorize the data. Towards this end, when the stop-ratio is determined to be smaller than a prespecified desired percentage *successively* for a given number of times (this is denoted by the metric "stop-num"), the algorithm will treat the error FVU as being stabilized, and hence will proceed to the next step to explore the possibility of further reducing the training error by extending a new layer to the network. It should be noted that the stop-ratio and the stop-num are to be generally determined by trial and error, and one may utilize prior experience derived from simulation results to assign them. Below, we provide some general guidelines for selecting these parameters.

*C*1: Based on our extensive simulation results, a rule of thumb, say 3–10 percent, should be reasonable for the stop-ratio. If a new hidden unit or layer is needed, then it is not difficult to observe that this bound will be easily surpassed. Generally speaking, having a network with a few hidden layers is clearly more advantageous over a network with many hidden layers. Therefore, this ratio may initially be set to be relatively small for the construction of the first hidden layer so that the algorithm is given an opportunity to solve the problem by constructing only a single hidden layer. On the other hand if the problem is not representable by a single hidden layer, then the ratio has to be increased to penalize and to prevent the creation of deep hidden layers. This incrementally adjusting or adapting the ratio, that is starting initially with small values and gradually increasing it, will achieve improved solutions if properly followed (the concept is very similar to the assignment of learning rates in BP networks).

*C*2: The value of stop-num greater than 3 generally imposes stricter emphasis on the error FVU stabilization, but that may be too conservative to allow redundant units to be added. Incentives can also be introduced for stop-num, similar to the stop-ratio case, by using larger values for the first hidden layer, say 4 or 5, and taking smaller ones for the subsequent layers, say 3 or even 2.

4.2. Convergence of the proposed algorithm

In [18], a new objective function or criterion for input-side weights training was derived as

$$J = \frac{(\sum_{j=1}^{P} e_{n-1}^{j} f_n(s_n^j))^2}{\sum_{j=1}^{P} (f_n(s_n^j))^2}, \tag{6}$$

where e_{n-1}^{j} is the output node tracking error for the hidden layer with $n-1$ hidden neurons and $f_n(s_n^j)$ is the output of the nth hidden unit for the jth input–output pattern. It was concluded that this function compared to the other objective functions used for input-side weights training of a OHL-FNN results in a somewhat better performance in solving certain regression problems [18]. Note that the above cost function was derived under the assumption that the output weights of the already-added hidden units do not change during the output-side training. We have found that in all the experiments we conducted this assumption is never valid. When output-side training is performed it causes tangible changes to all the output-side weights of the existing $n-1$ hidden units. Therefore, from both theoretical and practical points of view it is quite interesting to investigate what is the optimum objective function for input-side training, and under what condition(s) the proposed algorithm will converge? This subsection is dedicated to address these questions.

Although, the discussion here is presented for a general single hidden layer network, the results would be equally valid and can be generalized to multi-hidden layers as our proposed strategy is incrementally adding hidden layers and at any given time these results will apply to the existing hidden layer under construction and consideration.

Let the output weights of the hidden units be denoted by $\mathbf{V}_{n-1} = (v_0, v_1, v_2, \ldots, v_{n-1})^{\mathrm{T}}$ before the nth new hidden unit is added to the network. The change in $\mathbf{V}_{n-1}$ as a result of the output-side training after the nth new hidden unit is added is denoted by $\Delta\mathbf{V}_{n-1}$. The summed squared error of the output after the addition of the nth hidden unit is now given by

$$\begin{aligned} SSE_n &= \sum_{j=1}^{P} (d^j - y_n^j)^2 \\ &= \sum_{j=1}^{P} \{d^j - (\mathbf{V}_{n-1}^{\mathrm{T}} + \Delta\mathbf{V}_{n-1}^{\mathrm{T}})\mathbf{f}_{n-1}^{j} - v_n f_n(s_n^j)\}^2 \\ &= \sum_{j=1}^{p} (e_{n-1}^{j} - \mathbf{U}_n^{\mathrm{T}} \mathbf{f}_n^j)^2, \end{aligned} \tag{7}$$

where

$$\mathbf{U}_n = [\Delta\mathbf{V}_{n-1}^{\mathrm{T}}, v_n]^{\mathrm{T}}, \tag{8}$$

$$\mathbf{f}_n^j = [f_0, f_1(s_1^j), \ldots, f_{n-1}(s_{n-1}^j), f_n(s_n^j)]^{\mathrm{T}} \tag{9}$$

with $f_0 = 1$. After some algebraic manipulations, Eq. (7) may be written as

$$SSE_n = SSE_{n-1} - 2\mathbf{U}_n^{\mathrm{T}}\mathbf{Q}_n + \mathbf{U}_n^{\mathrm{T}}\mathbf{R}_n\mathbf{U}_n, \tag{10}$$

where

$$\mathbf{Q}_n = \sum_{j=1}^{P} e_{n-1}^{j}\mathbf{f}_n^{j}, \tag{11}$$

$$\mathbf{R}_n = \sum_{j=1}^{P} \mathbf{f}_n^{j}(\mathbf{f}_n^{j})^{\mathrm{T}}. \tag{12}$$

Minimization of the above SSE_n leads to the following *LS* solution:

$$\mathbf{U}_n = \mathbf{R}_n^{-1}\mathbf{Q}_n \tag{13}$$

with the minimum value given by

$$SSE_n^{\mathrm{opt}} = SSE_{n-1} - \mathbf{Q}_n^{\mathrm{T}}\mathbf{R}_n^{-1}\mathbf{Q}_n. \tag{14}$$

Obviously, maximizing the 2nd term in the RHS of the above equation will maximize the convergence in minimizing the SSE (or MSE). Therefore, this would provide an important condition that would guarantee the convergence of our constructive algorithm, namely provided that the input-side training for each hidden unit makes the 2nd term in the RHS of (14) positive, the network error will be decreasing monotonically as the training progresses.

It should also be noted that the objective function must be designed in such a way that the 2nd term in the above equation is positive and possibly maximized. Since this term is highly nonlinear, and the many previously proposed objective functions in [18] are related to it in rather complicated ways, one is generally resorted to a large number of simulations to determine the most suitable objective function that works best for a given problem. This fact emphasizes the difficulties one has to deal with as far as the input-side training is concerned.

Direct maximization of the 2nd term appears to be the most effective way to achieve the fastest convergence. Therefore, we propose that the optimum objective function may be given by

$$J_{\mathrm{input,opt}} = \mathbf{Q}_n^{\mathrm{T}}\mathbf{R}_n^{-1}\mathbf{Q}_n \tag{15}$$

which is different from the objective function J given by (6) and derived in [18]. Note that the above objective function is of a theoretical interest, as it will be difficult to implement it in actual input-side training due to the matrix inversion $\mathbf{R}_n^{-1}$. Also, note that the correlation between the output error and the output of the new hidden unit as indicated by (A.1) (refer to Appendix A) is included in $J_{\mathrm{input,opt}}$, and maximization of this correlation will contribute to making $J_{\mathrm{input,opt}}$ positive. If this condition can be ensured each time a new hidden unit is trained, the convergence of our proposed constructive algorithm will then be guaranteed.

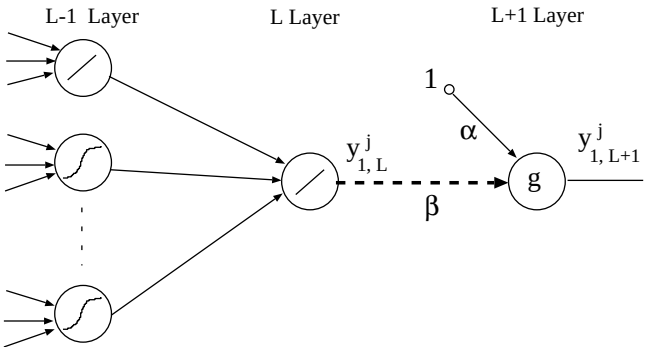


Fig. 2. Multi-hidden-layer network for the proposed strategy.

By denoting

$$\mathbf{Q}_n^{\mathrm{T}} = [\mathbf{Q}_{n-1}^{\mathrm{T}}, Q_n], \tag{16}$$

$$Q_n = \sum_{j=1}^{P} e_{n-1}^{j} f_n(s_n^j), \tag{17}$$

$$\mathbf{R}_n = \begin{bmatrix} \mathbf{R}_{n-1} & \mathbf{r}_{n-1} \\ \mathbf{r}_{n-1}^{\mathrm{T}} & r_n \end{bmatrix}, \tag{18}$$

$$\mathbf{r}_{n-1} = \sum_{j=1}^{P} f_n(s_n^j)\mathbf{f}_{n-1}^j, \tag{19}$$

$$r_n = \sum_{j=1}^{P} (f_n(s_n^j))^2 \tag{20}$$

and assuming that $\mathbf{r}_{n-1} = \mathbf{0}$ (implying that the output of the new hidden unit is not correlated with the outputs of all the previous hidden units, although this usually may not be satisfied in network training), we obtain

$$SSE_n^{\mathrm{opt}} = SSE_{n-1} - \mathbf{Q}_{n-1}^{\mathrm{T}} \mathbf{R}_{n-1}^{-1} \mathbf{Q}_{n-1} - Q_n^2 / r_n. \tag{21}$$

It follows that minimization of SSE_n yields the objective function J given by (6) and derived in [18, the 3rd term in the RHS of (2.6)]. Given that the assumptions stated earlier are not valid during the constructive learning process, the objective function J is therefore only "suboptimal". In other words, what we have derived here is an optimal objective function that does include the objective function derived in [18] as its special case. Implementing or developing a network structure that would realize the proposed objective function is left as a topic of future research.

Before the experimental results are presented, let us explain in more details why the proposed constructive algorithm cannot automatically be extended to classification problems where the output node has a nonlinear activation function. As shown in Fig. 2, suppose in the $(L-1)$th hidden layer there are n_{L-1} hidden units, and the

constructive algorithm is at the point of creating a new hidden layer by generating a new output node having the activation function $g(\cdot)$. Consequently, the following equation should hold:

$$y_{1,L+1}^{j} = g(\beta y_{1,L}^{j} + \alpha), \quad j = 1, 2, \ldots, P, \tag{22}$$

where β and α are the input-side weight and bias of the newly added output node, respectively. If we set $\beta = 1$ and $\alpha = 0$, and the activation function of the new output node is simply selected as linear, that is $g(x) = x$, we then obtain

$$y_{1,L+1}^{j} = y_{1,L}^{j}, \quad j = 1, 2, \ldots, P. \tag{23}$$

This implies that the training error from the previous output node will be passed on to the newly generated output node unchanged as is. When a new hidden unit is added to the Lth layer, the SSE at the new output node will be monotonically decreased.

On the other hand if $g(\cdot)$ is nonlinear, the parameters α and β should now satisfy the following expression:

$$\beta y_{1,L}^{j} + \alpha = g^{-1}(y_{1,L+1}^{j}) = g^{-1}(y_{1,L}^{j}) \tag{24}$$

for a given j. For serial learning or a single training sample, (24) may be satisfied without any difficulty (as we have one equation and two variables leading to non-unique solutions for α and β). For batch learning, which is the case being considered in this paper, we need to ensure that the following set of equations hold simultaneously:

$$\begin{aligned}
&\beta y_{1,L}^{1} + \alpha = g^{-1}(y_{1,L}^{1}),\\
&\beta y_{1,L}^{2} + \alpha = g^{-1}(y_{1,L}^{2}),\\
&\vdots\\
&\beta y_{1,L}^{P} + \alpha = g^{-1}(y_{1,L}^{P}).
\end{aligned} \tag{25}$$

Obviously, these simultaneous equations could have a unique solution only if P is 2. In most real-life problems the number of training samples is always larger than 2. Therefore, the best that one can accomplish from the above set of simultaneous equations is to work with the following least square (LS) solution, namely

$$\begin{bmatrix} \beta \\ \alpha \end{bmatrix} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}B, \tag{26}$$

where

$$A = \begin{bmatrix} y_{1,L}^{1} & 1 \\ y_{1,L}^{2} & 1 \\ \vdots & \vdots \\ y_{1,L}^{P} & 1 \end{bmatrix}, \quad B = \begin{bmatrix} g^{-1}(y_{1,L}^{1}) \\ g^{-1}(y_{1,L}^{2}) \\ \vdots \\ g^{-1}(y_{1,L}^{P}) \end{bmatrix}.$$

The summed square error of the above simultaneous equations is given by

$$J = \frac{1}{2}\sum_{j=1}^{P}\{\beta y_{1,L}^{j} + \alpha - g^{-1}(y_{1,L}^{j})\}^2$$

$$= \frac{1}{2}\left(A\begin{bmatrix}\beta\\ \alpha\end{bmatrix} - B\right)^{\mathrm{T}}\left(A\begin{bmatrix}\beta\\ \alpha\end{bmatrix} - B\right)$$

$$= \frac{1}{2}B^{\mathrm{T}}[I - A(A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}]B > 0. \tag{27}$$

Therefore, the output $SSE(L+1)$ at the new output node will be different from that of the previous output node, and is given by

$$SSE(L+1) = \sum_{j=1}^{P}\{d^j - g(\beta y_{1,L}^{j} + \alpha)\}^2, \tag{28}$$

where d^j is the desired target.

From the above analysis, it is clear now that the SSE may actually increase when a new output layer with a nonlinear node is added to the network. This is generally an undesirable consequence. On the other hand, if the addition of hidden units to the previous layer cannot only compensate the increase in SSE but also further reduce the SSE, the proposed algorithm may be extended to construct multilayer FNN with nonlinear activation functions, at least for some specific applications. The details regarding the conditions and these issues will not be pursued further in this paper.

5. Simulation results for regression problems

To confirm the effectiveness and potential of the proposed strategy, we have conducted several simulations as applied to regression problems. Unless explicitly specified, for all the simulation results shown below stop-ratio = 5% and stop-num = 3 during the entire construction process.

The first example considered is a linear regression function with noise given by

$$g(x) = 0.75x + 0.25 + v, \quad 0 < x < 1. \tag{29}$$

One hundred (100) samples are used for training. The SNR is 10 dB. Fig. 3 shows the training and the generalization error FVUs associated with the first and the second hidden layers. Fig. 3(a) indicates that the addition of sigmoidal hidden units to the first hidden layer resulted in little training FVU reduction, and the algorithm detected an error FVU saturation when the fourth hidden unit is added. This implies that all the three nonlinear hidden units added to the network are totally redundant and only the linear unit is needed, resulting in a linear network for representing a linear regression problem. To observe the possibility of reducing the FVU by using a deeper network,

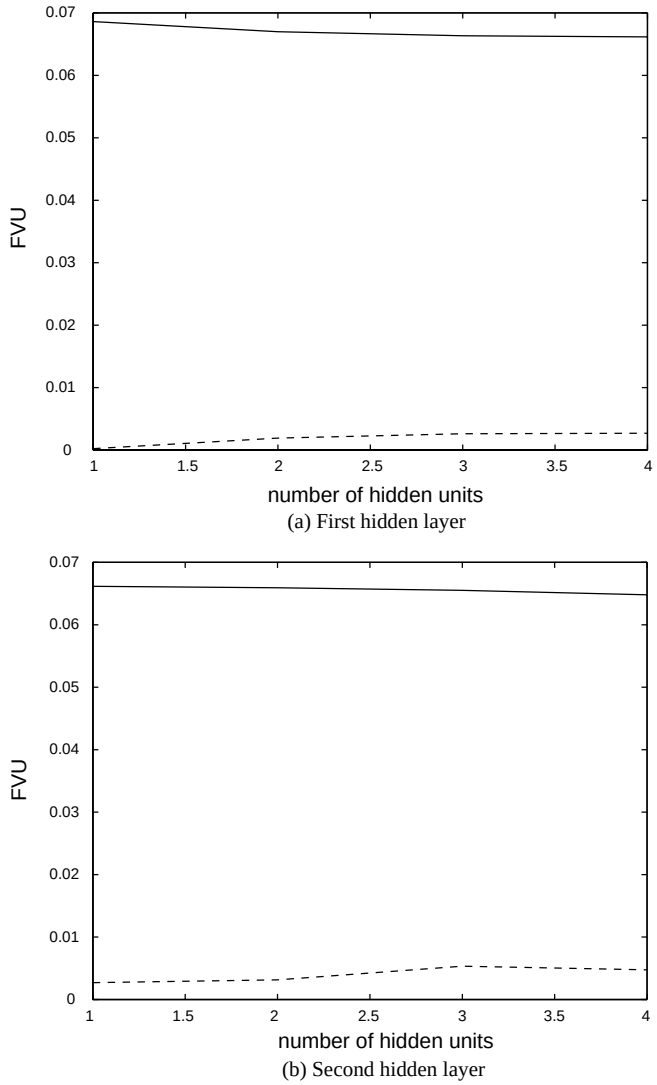
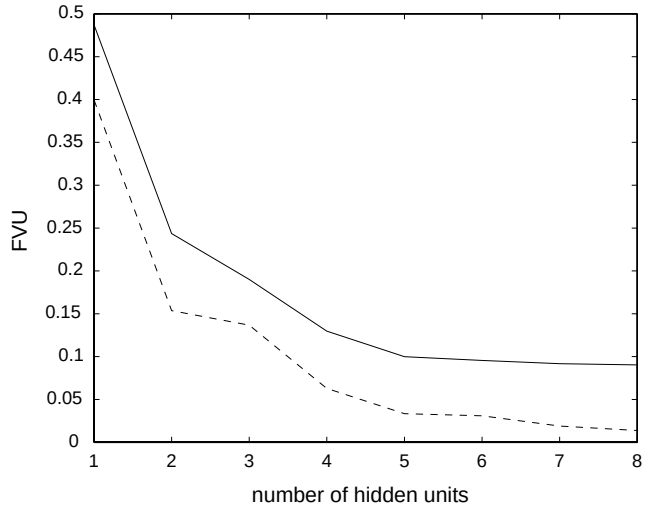


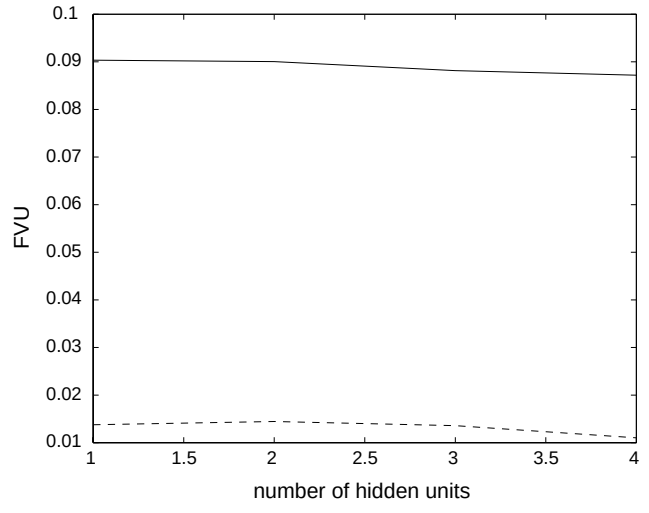
Fig. 3. Training (solid line) and generalization (dashed line) FVUs of a multi-hidden-layer network constructed for a simple linear regression problem. (a) First hidden layer; (b) Second hidden layer.

the second hidden layer was also added. Fig. 3(b) depicts the FVUs. Obviously, the saturation of the training FVU is more pronounced in this case.

The second example considered is a sinusoid with noise. One hundred (100) samples are used for training with the SNR selected to be 10 dB. The FVUs for different constructed hidden layers are depicted in Fig. 4. It follows from Fig. 4(a) that even though the first hidden layer reduces the training FVU very rapidly, the second hidden layer does contribute to the FVU reduction very minimally due to the performance



(a) First hidden layer



(b) Second hidden layer

Fig. 4. Training (solid line) and generalization (dashed line) FVUs of a multi-hidden-layer network constructed for a sinusoid with noise problem. (a) First hidden layer; (b) Second hidden layer.

saturation (stop-ratio = 5%). This implies that the second hidden layer is not required for this problem and simply a one hidden layer network may provide the best solution as provided by the proposed algorithm.

The third example considered is the 2-dimensional function "Complicated Interaction Function" (CIF) [9,18] given by

$$g(x_1,x_2) = 1.9(1.35 + e^{x_1} e^{-x_2} \sin(13(x_1 - 0.6)^2) \sin(7x_2)). \tag{30}$$

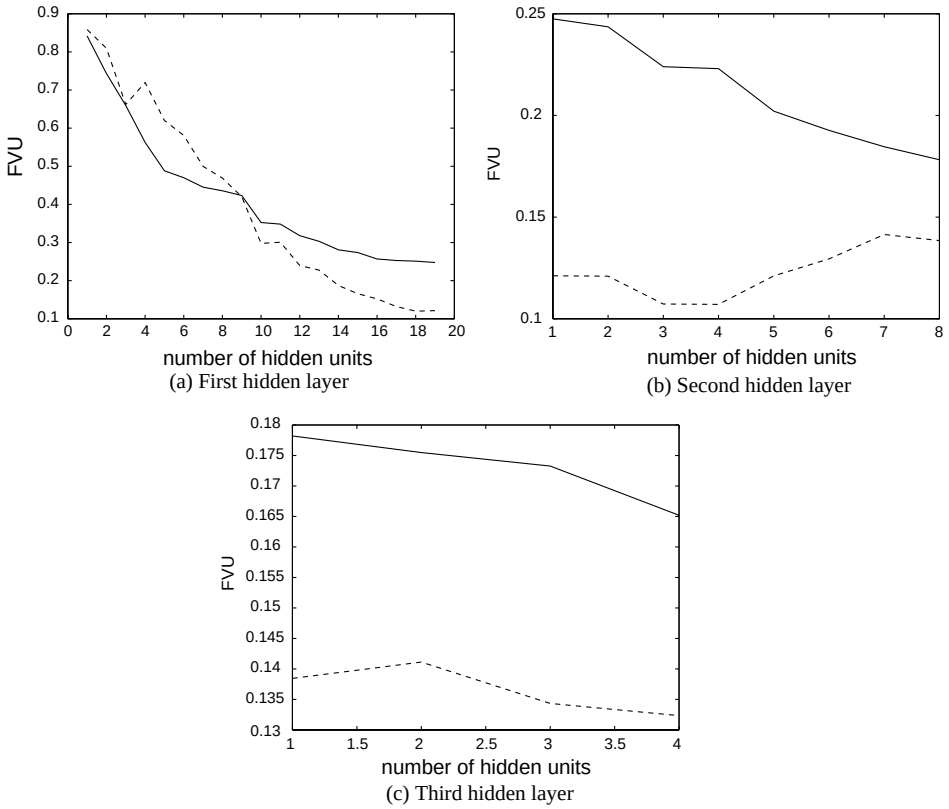


Fig. 5. Training (solid line) and generalization (dashed line) FVUs of a multi-hidden-layer network constructed for CIF. (a) First hidden layer; (b) Second hidden layer; (c) Third hidden layer.

Two hundred and twenty-five (225) uniformly distributed random points were generated from an interval $[0, 1]$ for network training. Ten thousand (10,000) uniformly sampled points from the same interval without additive noise were used to test the generalization performance of the trained network. The SNR is selected about 6 dB. Two hundred and twenty-five (225) data points are used to train a network. The results are given in Fig. 5. Clearly, it can be observed that the second hidden layer is needed in this case, however the third hidden layer does not contribute much to representing this problem (based on the stop-ratio of 5%). Therefore, a two-hidden-layer network with a small number of hidden units in the second hidden layer is all that is required for representing the CIF problem with noise. It may also be concluded from Fig. 5(b) that the OHL network with a sufficient number of hidden units may also yield satisfactory solution to the CIF problem, since the FVU associated with the second hidden layer saturates rather quickly. The generalized surfaces of CIF for different number of hidden units along with its original surface are shown in Fig. 6.

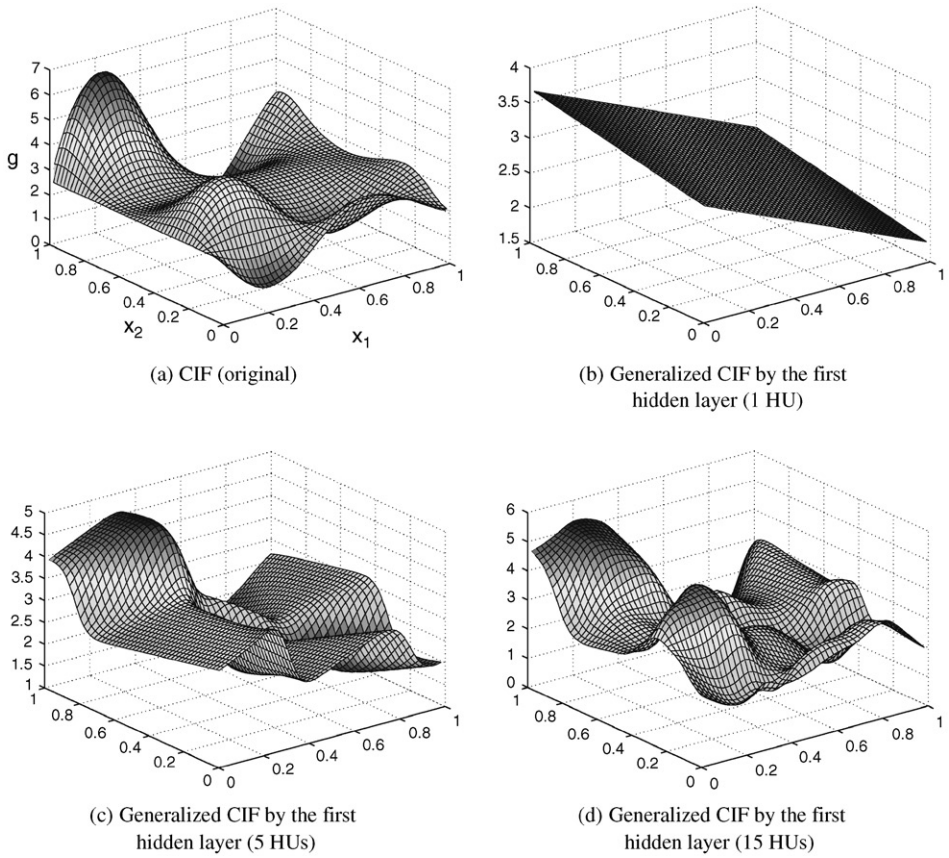


Fig. 6. Original and generalized CIF by the first hidden layer. HU denotes hidden unit. (a) CIF (original); (b) Generalized CIF by the first hidden layer (1 HU); (c) Generalized CIF by the first hidden layer (5 HUs); (d) Generalized CIF by the first hidden layer (15 HUs).

The last example we consider is a "Simple 3-dimensional Analytical Function" (SAF) [28] given by

$$g(x_1, x_2, x_3) = \frac{1}{1 + \mathrm{e}^{-\mathrm{e}^{x_1} + (x_2 - 0.5)^2 + 3\sin(\pi x_3)}}. \tag{31}$$

One thousand (1000) random samples with additive noise were generated for network training, and five thousand (5000) random samples without additive noise were generated for verifying the network generalization. Two hundred (200) samples are used to train a network. The SNR is selected to be 10 dB. The incentive given to the first hidden layer is by selecting its stop-ratio to 5%, and for the other hidden layers this parameter is set to 10%. Clearly, from Fig. 7 it follows that a two-hidden-layer network is all that is required for representing the SAF problem, as the 3rd layer causes the

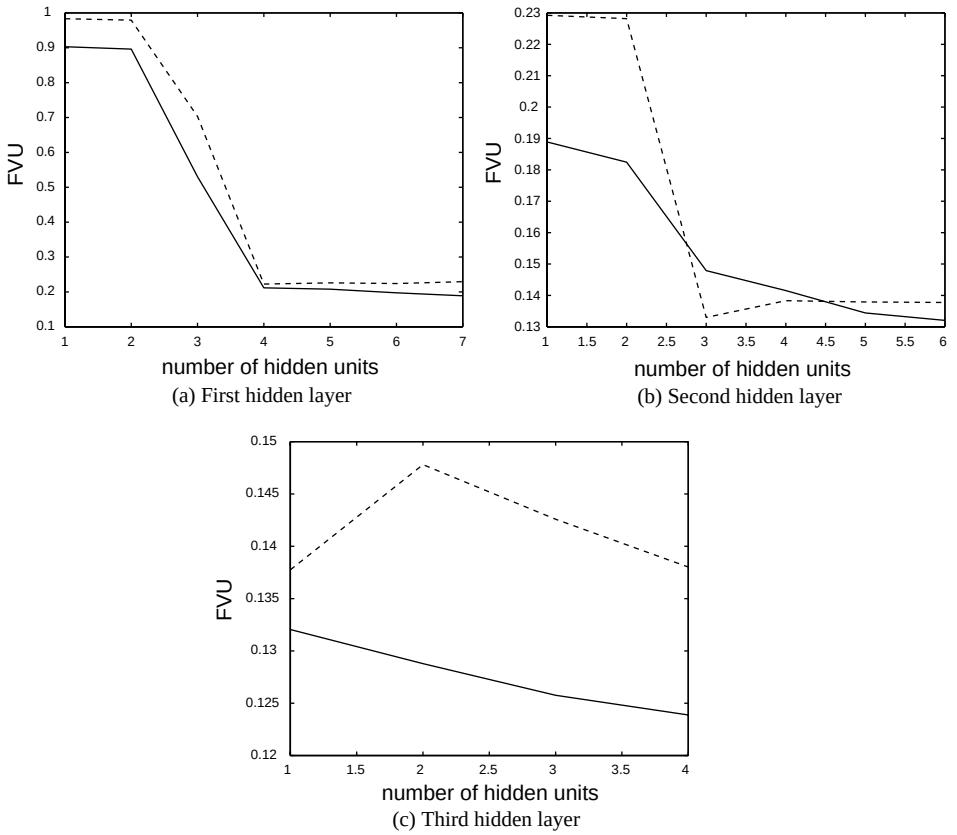


Fig. 7. Training (solid line) and generalization (dashed line) FVUs of a multi-hidden-layer network constructed for SAF. (a) First hidden layer; (b) Second hidden layer; (c) Third hidden layer.

stop-ratio to be satisfied each time a new hidden unit is added until the stop-num was reached.

To summarize, for the four examples given above we conclude that our proposed algorithm has constructed networks having simultaneously sufficient number of hidden layers and sufficient number of hidden units in each hidden layer, depending on the complexity of the problem that is being considered.

6. Discussion and conclusions

Research in developing techniques for the design and selection of neural network architectures suitable for solving classes of problems have been of interest for some time now. The design of network architectures may be viewed as a multi-criterion optimization problem where one is searching for a suitable network architecture to

be used for a class of application problems. Adaptive constructive neural networks as considered in this paper proceed through several optimization stages where neurons and layers are added to or pruned from a network until desired or required performance specifications are obtained. Alternatively, evolutionary algorithms [15] offer an attractive and relatively efficient alternative approach for the search of a near-optimal or sub-optimal solutions in a variety of problem domains. For example, in one approach a Genetic Algorithm (GA) and a Backpropagation network can be combined for training a neural network. In [36] the authors use a GA whose populations is a set of neural networks that have been initialized to different number of hidden layers and where a pruning method is also invoked to reduce the architecture of the oversized networks. Furthermore, in [6] evolutionary neural networks are proposed as a more efficient way of searching for the optimal set of learning parameters and topologies. This area of investigation has other interesting implications in facilitating possibly better understanding to solutions to other important open problems in artificial intelligence, cognitive modeling, among others.

In this paper, our focus has been concentrated on developing a new strategy for adaptively constructing a multi-hidden-layer feedforward neural network. The proposed algorithm extends the constructive algorithm developed for adding hidden units to a one-hidden-layer network by generating additional hidden layers. The network obtained by using our proposed algorithm has regular connections to facilitate hardware implementation. The constructed network adds new hidden units and layers incrementally only when they are needed based on monitoring the residual error that cannot be reduced any further by the already existing network. The advantages of our proposed algorithm are stated explicitly in Sections 3.2 and 4. It is demonstrated through the presented experimental results that the proposed algorithm is very effective as well as efficient in representing certain nonlinear problems. Further experimental results are needed for more complicated regression problems to determine the effectiveness of the new algorithm in other more complicated classes of nonlinear problems. It would be highly desirable to develop a more formal approach where the outcome of the optimized network structure becomes less dependent on the initialization of the parameters such as *stop-ratio* and *stop-num*, for example. Future work will also involve the application of the proposed method to several benchmarks as available in the UCI Machine Learning Database [1] such as the *Cancer1a* (Wisconsin breast cancer database) and *Glass* (classification of glass types).

It is well-known that the outcome of network training is strongly dependent on the choice of the initialization of the parameters of the network. Hence, when one repeats the training of the network with a different (randomly) selected initial conditions, there is a likelihood that the procedure may not necessarily yield the same network. Thus, we are encouraged to explore a variety of initial conditions if we are to achieve a more reliable network yielding a "best" minimum training error. This clearly is an important issue that needs further investigation.

On the other hand, it is conceivable that one may be able to do better by combining an ensemble, or a bank of networks that were trained during the above repetitive process (similar to a maximum selector concept), by choosing the single "best" network for a given set of input conditions.

Conducting further analysis, evaluation and comparison of evolutionary networks, genetic algorithms and the concepts presented in this article are topics for future research.

Acknowledgements

The authors would like to express their thanks to the reviewers for their constructive remarks and suggestions.

Appendix A. The "quickprop" algorithm

The *quickprop* algorithm developed by Fahlman [10] has played an important role in the input-side training of our constructive FNNs. In this appendix, a brief introduction to this algorithm is provided in the context of correlation-based input-side training. Note that *quickprop* is a second-order optimization method based on Newton's method. As shown below, it is simple in form, but has been found to be surprisingly effective when used iteratively [10].

The correlation-based objective function for input-side training is selected as follows (with the abuse of some previous notations):

$$J_{\text{input}} = \left| \sum_{j=1}^{P} (e^{j}_{n-1,o} - \bar{e}_{n-1,o})(f_n(s^j_n) - \bar{f}_n) \right| \tag{A.1}$$

with

$$\bar{e}_{n-1,o} = \frac{1}{P} \sum_{j=1}^{P} e^{j}_{n-1,o}, \tag{A.2}$$

$$\bar{f}_n = \frac{1}{P} \sum_{j=1}^{P} f_n(s^j_n), \tag{A.3}$$

$$s^j_n = \sum_{i=0}^{M} w_{n,i} x^j_i, \tag{A.4}$$

$$e^{j}_{n-1,o} = d^j - y^{j}_{n-1,o}, \tag{A.5}$$

where it is assumed that there are already $n-1$ hidden units in the network. The above objective function is used to train the input-side weight $\{w\}$ of the nth hidden unit, $f_n(\cdot)$ is the activation function of the nth hidden unit, x^j_i is the ith element of the input vector of dimension $M \times 1$, $e^{j}_{n-1,o}$ is the output error at the oth output layer due to the existing $n-1$ hidden units and $f_n(s^j_n)$ is the output of the nth hidden unit, all defined for the training sample j. The derivative of J_{input} with respect to an input-side

weight is given by

$$\frac{\partial J_{\text{input}}}{\partial w_{n,i}} = \sum_{j=1}^{P} \delta_j x_i^j, \tag{A.6}$$

$$\delta_j = \sum_{o=1}^{I} C_0 (e_{n-1,o}^j - \bar{e}_{n-1,o}) \frac{\mathrm{d} f_n(s_n^j)}{\mathrm{d} s_n^j},$$

$$C_0 = \operatorname{sgn}\left(\sum_{j=1}^{P} (e_{n-1,o}^j - \bar{e}_{n-1,o})(f_n(s_n^j) - \bar{f}_n) \right),$$

where for simplicity, $\bar{f}_n$ is treated as a constant in the above calculation, even though $\bar{f}_n$ is actually a function of the input-side weights. Let us define

$$S_{n,i}(t) = -\frac{\partial J_{\text{input}}}{\partial w_{n,i}}, \quad i = 0, 1, \ldots, M, \tag{A.7}$$

where t is the iteration step. The *quickprop* algorithm maximizing (A.1) may now be expressed by

$$\Delta w_{n,i}(t) = \begin{cases} \epsilon S_{n,i}(t), & \text{if } \Delta w_{n,i}(t-1) = 0, \quad (i = 0, 1, \ldots, M), \\ \dfrac{S_{n,i}(t) \Delta w_{n,i}(t-1)}{S_{n,i}(t-1) - S_{n,i}(t)}, & \text{if } \Delta w_{n,i}(t-1) \neq 0 \text{ and } \dfrac{S_{n,i}(t)}{S_{n,i}(t-1) - S_{n,i}(t)} < \mu, \\ \mu \Delta w_{n,i}(t-1), & \text{otherwise.} \end{cases} \tag{A.8}$$

where $\Delta w_{n,i}(t-1) = w_{n,i}(t) - w_{n,i}(t-1)$, and ϵ and μ are positive user-specified parameters. Note that in the simulation results presented in this work μ is selected as 1.75 and ϵ is selected as 0.35. Moreover, the nonlinear activation functions are all selected as "logsig".

References

[1] http://www1.ics.uci.edu/~mlearn/mlrepository.html.

[2] T. Ash, Dynamic node creation in backpropagation networks, Connection Sci. 1 (4) (1989) 365–375.

[3] N.K. Bose, P. Liang, Neural Network Fundamentals with Graphs, Algorithms, and Applications, McGraw-Hill, Inc., New York, 1996.

[4] W.L. Buntine, A.S. Weigend, Bayesian backpropagation, Complex Syst. 5 (1991) 603–643.

[5] G. Castellano, A.M. Fanelli, M. Pelillo, An iterative pruning algorithm for feedforward neural networks, IEEE Trans. Neural Networks 8 (3) (1990) 519–531.

[6] P.A. Castillo, J. Gonzalez, J.J. Merelo, V. Rivas, G. Romero, A. Prieto, Sa-prop: optimization of multilayer perceptron parameters using simulated annealing, Lecture Notes in Computer Science, Vol. I, Vol. 1606, Springer, Berlin, 1998, 661–670.

[7] Y. Chauvin, A back-propagation algorithm with optimal use of hidden units, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing (2), Denver, Morgan Kaufmann, San Mateo, CA, 1990, pp. 642–649.

[8] Y. Le Cun, J.S. Denker, S.A. Solla, Optimal brain damage, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing (2), Denver, Morgan Kaufman, San Mateo, CA, 1990, pp. 598–605.
[9] T. Draelos, D. Hush, A constructive neural network algorithm for function approximation, in: IEEE International Conference on Neural Network, Vol. 1, Washington, DC, 1996, pp. 50–55.
[10] S.E. Fahlman, C. Lebiere, The cascade-correlation learning architecture, Tech. Rep., CMU-CS-90-100, Carnegie Mellon University, 1991.
[11] W. Fang, R.C. Lacher, Network complexity and learning efficiency of constructive learning algorithms, in: Proceedings of the International Conference on Artificial Neural Networks, 1994, pp. 366–369.
[12] D.R. Hush, B.G. Horne, Progress in supervised neural networks, IEEE Signal Process. Mag. 10 (1) (1993) 8–39.
[13] M. Ishikawa, A structural learning algorithm with forgetting of link weights, Tech. Rep. TR-90-7, Electrotechnical Lab., Tsukuba-City, Japan, 1990.
[14] E.D. Karnin, A simple procedure for pruning back-propagation trained neural networks, IEEE Trans. Neural Networks 1 (2) (1990) 239–242.
[15] J.R. Koza, Genetic Programming: on the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.
[16] T.Y. Kwok, D.Y. Yeung, Bayesian regularization in constructive neural networks, in: Proceedings of the International Conference on Artificial Neural Networks, Bochum, Germany, 1996, pp. 557–562.
[17] T.Y. Kwok, D.Y. Yeung, Constructive algorithms for structure learning in feedforward neural networks for regression problems, IEEE Trans. Neural Networks 8 (3) (1997) 630–645.
[18] T.Y. Kwok, D.Y. Yeung, Objective functions for training new hidden units in constructive neural networks, IEEE Trans. Neural Networks 8 (5) (1997) 1131–1148.
[19] T.C. Lee, Structure Level Adaptation for Artificial Neural Networks, Kluwer Academic Publishers, Dordrecht, 1991.
[20] C.T. Leondes, Neural Network Systems Techniques and Applications: Algorithms and Architectures, Academic Press, New York, 1998.
[21] R. Lippmann, An introduction to computing with neural nets, IEEE ASSP Mag. 4 (2) (1987) 4–22.
[22] G.D. Magoulas, M.N. Vrahatis, G.S. Androulakis, Efficient backpropagation training with variable stepsize, Neural Networks 10 (1) (1997) 69–82.
[23] M.C. Mozer, P. Smolensky, Skeletonization: a technique for trimming the fat from a network via relevance assessment, in: D.S. Touretzky (Eds.), Advances in Neural Information Processing (1), Denver, Morgan Kaufman, San Mateo, CA, 1989, pp. 107–115.
[24] T.M. Nabhan, A.Y. Zomaya, Toward generating neural network structures for function approximation, Neural Networks 7 (1) (1994) 89–99.
[25] S. Osowski, P. Bojarczak, M. Stodolski, Fast second-order learning algorithm for feedforward multilayer neural networks and its applications, Neural Networks 9 (9) (1996) 1583–1596.
[26] L. Prechelt, Investigation of the Cascor family of learning algorithms, Neural Networks 10 (5) (1997) 885–896.
[27] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart, J.L. McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press, Cambridge, MA, 1986, pp. 318–362.
[28] T. Samad, Backpropagation with expected source values, Neural Networks 4 (1991) 615–618.
[29] D. Sarkar, Methods to speed up error back-propagation learning algorithm, ACM Comput. Surveys 27 (4) (1995) 519–542.
[30] R. Setiono, L.C.K. Hui, Use of a quasi-Newton method in a feedforward neural network construction algorithm, IEEE Trans. Neural Networks 6 (1995) 273–277.
[31] A.J. Shepherd, Second-Order Methods for Neural Networks, Springer, London, 1997.
[32] F. Stager, M. Agarwal, Three methods to speed up the training of feedforward and feedback perceptrons, Neural Networks 10 (8) (1997) 1435–1443.
[33] I.V. Tetko, Efficient partition of learning data sets for neural network training, Neural Networks 10 (8) (1997) 1361–1374.
[34] H.H. Thodberg, A review of Bayesian neural networks with an application to near infrared spectroscopy, IEEE Trans. Neural Networks 7 (1996) 56–72.

[35] W. Weng, K. Khorasani, An adaptive structural neural network with application to eeg automatic seizure detection, Neural Networks 9 (7) (1996) 1223–1240.
[36] X. Yao, Y. Liu, Towards designing artificial neural networks by evolution, Appl. Math. Comput. 91 (1) (1998) 83–90.

Liying Ma received the B.S. degree from Northeastern University, Shengyang, China, in 1984, the M.S. degree from Hiroshima University, Higashi-Hiroshima, Japan, in 1991 and the Ph.D. degree from Concordia University, Montreal, Canada, in 2001 in Electrical and Computer Engineering. Dr. Ma was with the Automation Research Institute of Ministry of Metallurgical Industry, Beijing, China, as a research Electrical Engineer from 1984 to 1988. She worked as a systems engineer at Yokogawa Engineering Service Co. Tokyo, Japan, before pursuing her Ph.D. studies in Canada. Her main areas of research interests are in neural networks and their applications to image processing and pattern recognition.

K. Khorasani received the B.S., M.S., and Ph.D. degrees in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign in 1981, 1982 and 1985, respectively. From 1985 to 1988 he was an Assistant Professor at the University of Michigan at Dearborn and since 1988, he has been at Concordia University, Montreal, Canada, where he is currently a Professor in the Department of Electrical and Computer Engineering and since 1998 an Associate Dean in the Faculty of Engineering and Computer Science. His main areas of research are in stability theory, nonlinear and adaptive control, modeling and control of flexible link/joint manipulators, neural network applications to pattern recognition, robotics and control, adaptive structure neural networks, and distributed and collaborative force feedback of haptic interfaces in virtual environments. He has authored/co-authored over 175 publications in these areas.