INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Constructive Neural Networks with Applications to Image Compression and Pattern Recognition

Liying Ma

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

May 2001

ABSTRACT

Constructive Neural Networks with Applications to Image Compression and Pattern Recognition

Liying Ma, Ph.D.

Concordia University, 2001

The theory of Neural Networks (NNs) has witnessed a striking progress in the past fifteen years. The basic issues, such as determining the structure and size of the network, and developing efficient training/learning strategies have been extensively investigated.

This thesis is mainly focused on constructive neural networks and their applications to regression, image compression and pattern recognition problems. The contributions of this work are as follows. First, two new strategies are proposed for a constructive One-Hidden-Layer Feedforward NN (OHL-FNN) that grows from a small initial network with a few hidden units to one that has sufficient number of hidden units as required by the underlying mapping problem. The first strategy denoted as error scaling is designed to improve the training efficiency and generalization performance of the OHL-FNN. The second strategy is a pruning criterion that produces a smaller network while not degrading the generalization capability of the network.

Second, a novel strategy at the structure level adaptation is proposed for constructing multi-hidden-layer FNNs. By utilizing the proposed scheme, a FNN is obtained that has sufficient number of hidden layers and hidden units that are required by the complexity of the mapping being considered.

Third, a new constructive OHL-FNN at the functional level adaptation is developed. According to this scheme, each hidden unit uses a polynomial as its activation function that is different from those of the other units. This permits the

growing network to employ different activation functions so that the network would be able to represent and capture the underlying map more efficiently as compared to the fixed activation function networks.

Finally, the proposed error scaling and input-side pruning techniques are applied to regression, still and moving image compression, and facial expression recognition problems. The proposed constructive algorithm for creating multilayer FNNs is applied to a range of regression problems. The proposed polynomial OHL-FNN is utilized to solve both regression and classification problems. It has been shown through extensive simulations that all the proposed techniques and networks produce very promising results.

ACKNOWLEDGEMENTS

There are so many people who have given me support and help during my study at Concordia. First of all, I wish to extend my most heart felt thanks and sincere appreciations to my thesis advisor, Professor K. Khorasani, whose inspiration, guidance, advice, encouragement, patience and all of the support throughout my Ph. D. study have made this day possible. I am especially grateful for the constant/unfailing encouragement he has given me. I would not have completed this study without his expert guidance and substantial time and efforts. My special thanks go to Professor M.R. Azimi-Sadjadi for generously agreeing to serve as an external examiner of my thesis and for providing valuable comments on my thesis. Special thanks go to Professor W. Lynch who emphasized the need for rigor and precision in my thesis through tireless reviews of my dissertation document and suggestive discussions. Special thanks also go to Professor H. Poorooshasb and Professor F. Khendek, for reviewing my thesis and giving their valuable comments, as well as serving on my defense examination committee. Thanks also go to Professor R. Zmeureanu for chairing on my defense examination committee.

I would like to thank Dr. N. Kasabov for his stimulating discussions at international conferences.

I would also like to thank my friends Z. Xu, M. Wang, Y. Jiang, L. Chen, N. Guo and H. Deng for their friendship which accompanied me throughout my studies at Concordia. Particularly, I thank my friends J. Zan and Dr. Y. Wang for helping me deal with the details in arranging my defense.

This thesis is dedicated to my family for their endless love and support. My parents have given of themselves without any reservation. My brothers have always been there and wishing me the best. My husband has given me his understanding and all of his support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

b_z	bias of the hidden unit(s)
c_m	similarity of an image block to an image
$c_{opt}(\cdot)$	optimal compressor characteristic
$d, \mathbf{D}$	target for network training
e	output error of the NN
$e_{n-1,min}$	user-specified minimum value of output error
$e_{n-1,max}$	user-specified maximum value of output error
f_1, f_2, f_n, f	activation function of the hidden or the output node
f_{min}, f_{max}	user-specified minimum and maximum values for activation functions
$g(\cdot)$	regression map (function)
$h_n(\cdot)$	Hermit orthonormal polynomial
$H_n(\cdot)$	Hermit orthogonal polynomial
$\mathbf{H}(t)$	approximate Hessian inverse at iteration t
I	dimension of input vector to or number of output nodes of the NN
$\mathbf{I}$	identity matrix of proper dimension
J, J_{input}	objective function for input-side training
J_{output}	objective function for output-side training
L	number of hidden layers or horizontal/vertical size of an image
L_q	number of quantization levels ($= 2^R$)
M	dimension of the input vector to the NN
M_b	size of square block of an image
N	number of output nodes of the NN
P	number of training samples

$P_n(\cdot)$	other polynomials
$\mathbf{p}_n$	vector for square image block
$p_x(\cdot)$	probability density function
q	quantization error
$\mathbf{q}_n$	vector for square image block
$Q(\cdot)$	quantizer characteristic
$\mathbf{Q}$	cross-correlation matrix
R	bite rate (bits/pixel)
$\mathbf{R}$	auto-correlation matrix
s	input to a hidden unit
sgn	sign function
S	similarity between two images
S^A, S^B	sensitivity functions used for input-side pruning
U	gray-level of a pixel
U_{max}	largest possible gray level of a pixel in an image
v	output-side weight of a hidden unit
$\mathbf{v}$	weight matrix of the NN
V_1, V_2	variances of images
$V_{\mathbf{p}_n}, V_{\mathbf{q}_m}$	variances of image blocks
$\mathbf{w}$	weight matrix of the NN
x	input to the NN, input to a quantizer
$\mathbf{X}, \mathbf{x}$	input vector to the NN or predictor variable(s) of the regression map (function)
x_{max}	maximum value of decision level
x_k, y_k	decision and representation levels of a quantizer
y	output of the NN, output of a quantizer
Y	output or response of a regression map (function)
z	output of the hidden unit(s)

$\mathbf{Z}$	digital image
α	bias and weight of a node or a constant
α_n	coefficient of the Hermit orthonormal polynomial
β	weight of a node
$\gamma(t)$	difference gradient vector
$\delta(t)$	difference output weight vector
Δ	quantization step size
$\nabla J_{output}(\cdot)$	gradient of J_{output} with respect to output-side weights
ϵ	additive noise
ε	positive user specified parameter for *quickprop*
η	percentage of the number of connections or weights pruned
λ	a constant
μ	positive user specified parameter for *quickprop*
μ_1, μ_2	mean values of images
$\mu_{\mathbf{p}_n}, \mu_{\mathbf{q}_m}$	mean values of image blocks
ρ	compression ratio
$\sigma_{e_{n-1}}$	standard deviation of the output error
$\phi(\cdot)$	weighting function of the Hermit orthonormal polynomial

Chapter 1

Introduction

1.1 Background

Artificial neural networks (ANNs) or simply neural networks have been studied for more than 40 years to achieve human-like performance in real-life applications. Biologically inspired, ANNs are composed of many nonlinear computational elements operating in parallel and organized in a way that may or may not be related to the actual structure of the brain. Neurons or computational elements or nodes are connected by weights that are adjusted to achieve the application requirements. Figure 1.1 shows a simple neural network.

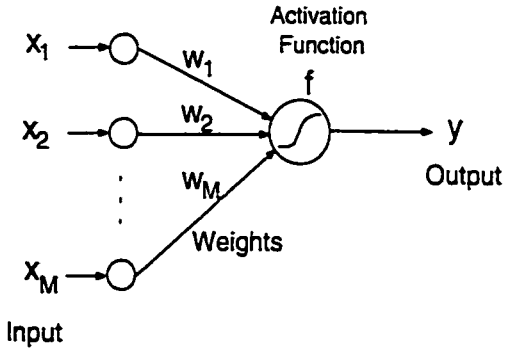


Figure 1.1: A simple neural network structure.

ANNs are characterized by their learning and generalization capabilities which are similar to the ones of the brain's. They learn from experience and can modify their performance by adjusting their behavior to respond to their environment. Many kinds of training algorithms have been developed. Once trained, they can generalize from previous examples to new ones due to their structures and their training strategies. Generalization ability of a trained network is critical, because it measures how well the network works when presented with a new input data or patterns that the network has not seen before.

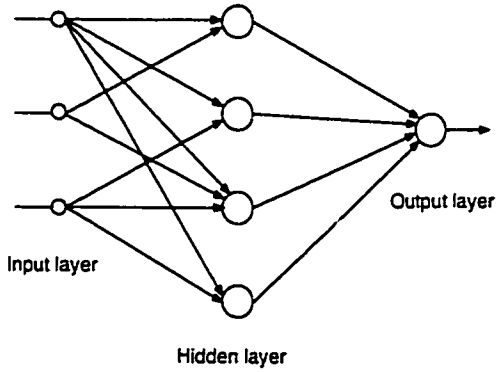
An ANN is specified by its structure, node activation function and learning or training rules/algorithms. The training rules specify the initial network weights or even initial structure, and decide how the network (weights and/or structure) should be adjusted to satisfy the prespecified performance requirements.

The theory of Neural Networks (NNs) has witnessed a striking progress in the past fifteen years since its resurgence in the early 1980s. The basic issues, such as determining the structure and size of a NN, developing an efficient NN learning strategy, and other related problems have been extensively investigated and a large number of NN topologies and training algorithms have emerged in the literature [1]-[7], [14]-[16].

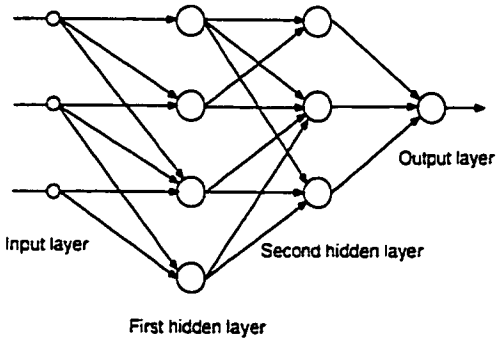
Furthermore, NNs have also found a vast number of real-life applications, such as in digital communications [8] (also see the references therein), control [9, 10], signal processing [11, 12], function approximations, regressions, pattern recognitions [6, 7, 15, 17], etc. The range of NN-based techniques for real-life applications is still expanding and is growing.

Among the numerous NNs paradigms, such as Hopfield networks, Hamming networks, Kohonen's self-organizing feature maps (SOM), Radial Basis Function (RBF) networks, and Multilayer Perceptrons (MLP), the feedforward NNs (FNNs) are the most popular due to their flexibility in structure, good representational capabilities, and large number of available training algorithms [1]-[7], [14, 15]. In

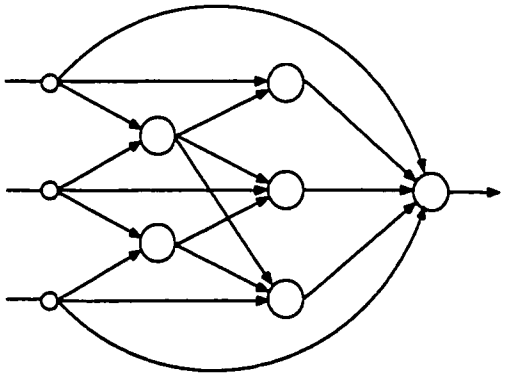
this thesis we will be mainly concerned with the FNNs. Some examples of FNNs [7] are shown in Figure 1.2.



(a) One hidden layer network (1HL)



(b) Multi-hidden layer network (MHL)



(c) Nonlayered with shortcut

Figure 1.2: Examples of FNN.

When using a NN, one needs to address three important problems. The solutions to these problems will significantly influence the overall performance of the NN as far as the generalization performance of the network to new data sets that have not been presented during the network training.

The first problem is the selection of data/patterns for network training. This is a problem that has practical concerns and has not received as much attention by researchers as the other problems. The training data set selection can have considerable effects on the performances of the trained network. Some research on this issue has been conducted in [13] (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 are developed in the literature, such as the classical error backpropagation (BP) [18] and its many variants [14], [19], [20] and the second-order algorithms [21, 22], to name a few. Many new training algorithms with faster convergence properties and less computational load are being developed every year by researchers in the NN community.

The third problem is the determination of the network architecture. 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 problems here are to find a network structure as small as possible to meet certain desired performance requirements, as well as the selection of activation function for the network nodes. What is usually done in practice is that the developer trains a number of networks with different sizes and activation functions, and then the smallest network that can fulfill all or most of the required performance specifications is selected. This amounts to a tedious process of trial and error that seems to be unfortunately unavoidable. The selection of activation functions for the network nodes can be considered as a part of network structure. Because the selection of activation functions actually makes the NN-based solution finding more complex, the log-sigmoidal or tan-sigmoidal functions are generally used. But there is no guarantee that they are the best choice. This thesis focuses on developing a systematic procedure for an automatic determination and/or adaptation of the network architecture and activation functions 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. Neural network is one of the most promising tools to address this problem by mapping the input data into the output data. 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 to address curve fitting problems. Generally acceptable results can not be achieved if too few coefficients are used, since the characteristics or features of the underlying function can not 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 size of the NN to the complexity of the problem. Obviously, algorithms that can determine an appropriate network architecture and/or size 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 [2], [34] (and the references therein]. Towards this end, in the next section, we review three general methods that deal with the problem of NN structure determination.

1.2 Review of adaptive structure neural networks

In this section, we give a brief review for three types of algorithms for adaptive structure neural networks, before we review the constructive feedforward NNs in detail in next section. These three types of algorithms are pruning algorithms, regularization and constructive learning algorithms.

1.2.1 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 determine 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 which estimates which units are "least important" and deletes them during training. Karnin [24] defined a sensitivity of the error function being minimized with respect to the removal of each connection and pruned the connections that show low sensitivity. Le Cun et al. [25] 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. [26] 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 [23]-[25]. The above pruning algorithms have the following limitations: (i) the size of the initial network may be difficult to determine *a priori*, and (ii) the computational cost are very excessive due to the fact that repeated pruning processes have to be performed.

1.2.2 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 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 based on a sensitivity criterion is one possible approach to deal with this situation. Alternatively, a scheme known as regularization, does impose some conditions on the NN training so that the unnecessary weights will be forced to converge to zero. This can be done by adding a penalty term to the error objective/cost function that is being minimized. Many different penalty terms are proposed in the literature that have been found to be effective for many problems, see for example the references [27]-[30].

However, the regularization techniques still can not determine the size of the network. The network size has to be determined in advance by the user. Furthermore, in the objective/cost function one has to weight appropriately the error term and the penalty term. 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 [31]-[33].

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.

1.2.3 Constructive learning algorithms

The third approach for determining the network size is known as constructive learning. Constructive learning alters the network structure simultaneously 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 adds additional hidden units and/or hidden layers until some prespecified error requirement is reached or no performance improvement can be observed. The network that is obtained in this way is a "reasonably" sized one for the given problem at hand. Generally, a "minimal" network is seldom achieved by using this strategy, however a "sub-minimal" network can be expected [34, 45]. This problem has attracted a lot of attention by several researchers and many promising algorithms have been proposed. An excellent work by Kwok and Yeung [34] surveys the major constructive algorithms in the literature. Dynamic node creation algorithm and its variants [35]-[39], activity-based structure level adaptation [40, 41], Cascade-Correction (CC) algorithms [42]-[44], and the constructive one-hidden-layer (OHL) algorithms [45, 46] are among the most important constructive learning algorithms developed so far.

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

A1. It is relatively 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 initial network size should be. Therefore, an initial

network that is much larger than actually required by the underlying problem is usually chosen in pruning algorithms, leading to a costly network training process.

A2. Constructive algorithms tend to build small networks due to their incremental learning nature. Networks are constructed that correspond to the complexity of the given problem and the specified performance requirements, while overly large efforts may be spent to trim the unnecessary weights contained in the network in pruning algorithms. Thus, constructive algorithms are generally more economical (in terms of training time and network complexity/structure) than pruning algorithms.

A3. In pruning algorithms and regularization-based techniques, several problem-dependent user parameters need to be properly specified or selected in order to obtain an "acceptable" and "good" network yielding satisfactory performance. This requirement makes these algorithms more difficult to be used in real life applications. On the other hand, constructive algorithms with incremental procedures are not affected by these drawbacks and problems.

In the next section a more detailed formulation of the constructive FNN as well as the description of the previous algorithms are provided.

1.3 Constructive algorithms for feedforward neural networks (FNNs)

In this section, we first give a simple formulation of the problem in training a constructive one-hidden-layer FNN in the context of nonlinear optimization. An introduction to and review of several existing constructive algorithms are provided in some detail, as a preamble and review to the chapters that follow. The advantages and drawbacks of these constructive algorithms are indicated and discussed.

Finally, the fundamental and the main concerns and focus of this thesis work will be provided.

1.3.1 Formulation of constructive FNN training

Suppose that a FNN is used to approximate a regression function whose input vector (or predictor variables) is indicated by a vector $\mathbf{X}$ of multi-dimension and its output (or response) is expressed by Y. Without loss of generality, we assume here that Y is a one-dimensional variable. A regression surface (input-output function) $g(\cdot)$ describes the relationship between $\mathbf{X}$ and Y. A FNN is trained and used to realize or represent this relationship. The input samples are $\mathbf{X} = (\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^P)$, and the corresponding target samples (or observations) are $\mathbf{D} = (d^1, d^2, \cdots, d^P)$ which are the output data contaminated by an additive white noise vector $\mathbf{\Lambda} = (\epsilon^1, \epsilon^2, \cdots, \epsilon^P)$, with $d^j = y^j + \epsilon^j$ for $j = 1, 2, \cdots, P$, where P is the number of patterns in the data set. The network training can be formulated as the following unconstrained least square (LS) nonlinear optimization problem

$$\min_{L,\mathbf{n},\mathbf{f},\mathbf{w}} \sum_{j=1}^{P} (d^j - y_L^j)^2 \tag{1.1}$$

subject to

$$\begin{cases} \mathbf{y}_1^j = f_1(\mathbf{w}_1 \mathbf{x}^j), \quad \mathbf{w}_1 \in \Re^{n_1 \times M}, \ \mathbf{y}_1^j \in \Re^{n_1}, \ \mathbf{x}^j \in \Re^M, \\ \mathbf{y}_2^j = f_2(\mathbf{w}_2 \mathbf{y}_1^j), \quad \mathbf{w}_2 \in \Re^{n_2 \times n_1}, \ \mathbf{y}_2^j \in \Re^{n_2}, \\ \vdots \\ y_L^j = f_L(\mathbf{w}_L \mathbf{y}_{L-1}^j), \quad \mathbf{w}_L \in \Re^{1 \times n_{L-1}}, \ y_L^j \in \Re^1 \end{cases} \tag{1.2}$$

where $\mathbf{n} = (n_1, n_2, \cdots, n_{L-1})$ is a vector for the number of hidden units at each hidden layer, $\mathbf{f} = (f_1, f_2, \cdots, f_L)$ denotes the activation function of each layer, f_1, f_2, $\cdots$, f_{L-1} are usually nonlinear activation functions, where f_L is linear for a regression problem, and $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_L)$ represents the weight matrix at each

layer (see Figure 1.3 for the structure of an MHL-FNN). All levels of adaptation are included in the above LS nonlinear optimization. The performance index (1.1) appears to be too complicated to be solved by any existing optimization technique to reach an acceptable solution due to the huge optimization space. Therefore, if one fixes some of the variables, the problem will become easier to solve. However, only a suboptimal solution can then be usually provided. Fortunately, it turns out that a suboptimal solution suffices in many practical situations.

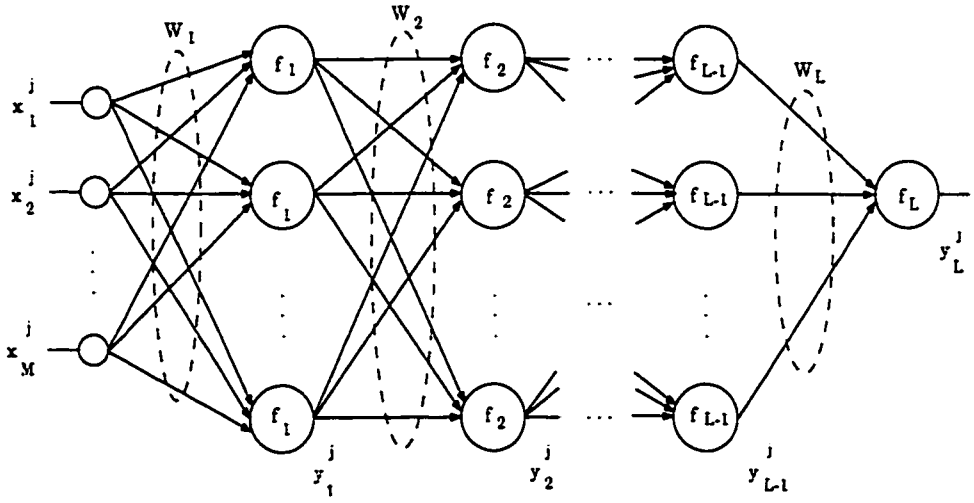


Figure 1.3: Structure for a multi-hidden-layer constructive FNN.

If on the other hand a OHL network is used to approximate the regression problem, its training is reduced to the following LS optimization problem

$$\min_{f_1,f_2,n_1,\mathbf{w}_1,\mathbf{w}_2} \sum_{j=1}^{P} (d^j - y_2^j)^2 \tag{1.3}$$

subject to

$$\begin{cases} \mathbf{y}_1^j = f_1(\mathbf{w}_1\mathbf{x}^j), \quad \mathbf{w}_1 \in \Re^{n_1 \times M}, \ \mathbf{y}_1^j \in \Re^{n_1}, \ \mathbf{x}^j \in \Re^M, \\ y_2^j = f_2(\mathbf{w}_2\mathbf{y}_1^j), \quad \mathbf{w}_2 \in \Re^{1 \times n_1}, \ y_2^j \in \Re^1. \end{cases} \tag{1.4}$$

It is not difficult to observe that even the above reduced LS nonlinear optimization problem is still not trivial to solve. This is partly due to the freedom in the selection of the activation functions and the number of hidden units that make the search for the solution to the optimization problem difficult to determine.

Actually, one can solve (1.1)-(1.2) and (1.3)-(1.4) only in an incremental fashion. That is, given certain variables, say the activation functions and the number of hidden layers and units, according to a systematic prespecified policy, solve the resulting LS optimization problem with respect to the remaining variables, and repeat the process until an acceptable solution or network is obtained. In this way, for a given choice of activation functions and the number of hidden layers and units, (1.1) and (1.3) reduce to two nonlinear optimization problems with respect to the weight matrices. These problems may be solved using the steepest-decent (for example, the BP) type or second-order (Quasi-Newton) recursive methods. The error surfaces represented by (1.1) and (1.3) have, in general, many local minima and the solution to the problem is sought for only one local minimum at a time. The error surfaces change their shapes each time the activation functions and/or the number of hidden layers and/or units are modified, resulting in different local minimum each time.

The four major constructive algorithms that are briefly reviewed below have adopted the above approach and can therefore provide only suboptimal solutions to (1.1) and (1.3). The constructive OHL-FNNs architecture developed in Chapters 2 and 4 also provide suboptimal solutions to (1.3) in one way or another, whereas the constructive multilayer FNN proposed in Chapter 3 provides a suboptimal solution to (1.1).

1.3.2 Dynamic node creation (DNC)

When one attempts to use a FNN to solve a mapping problem, it is generally very difficult to obtain *a priori* the proper size for the FNN. One may be given some underlying information on the training data, however the information may not be easily and directly used to assist in determining a good "guess" for the network size. A simple and an intuitive idea would be to start training from a rather small network, enlarging it when needed to automatically obtain a network that would represent the complexity of the problem being considered within a prespecified performance requirements.

Dynamic node creation (DNC) developed by Ash [35] and its variants [36]-[39] are constructive algorithms that are based on this intuition. In these algorithms, the training process begins with a small network. The backpropagation algorithm, Newton's method, Quasi-Newton methods, recursive least-squares algorithms, and the gradient-based algorithms are then used for training the weights of the network. When the training error is stabilized to some prespecified range, the training process is stopped. New hidden units (nodes) are added to the stabilized network one at a time, and the enlarged network is trained again. This process will be continued until a desired error bound is reached or some stopping conditions are satisfied. Figure 1.4 shows a typical FNN structure under construction using DNC.

The major advantage of these algorithms is the simplicity of the node creation policy. Any training algorithm for a fixed structure network can be used. The convergence of the algorithms to the target function follows directly from the universal approximation property of the FNN structure [18]. However, the major difficulty with these algorithms is the significant increase in the computational load of the network training when the network becomes "significantly" large. This is due to the fact that the enlarged network has to be retrained each time a new hidden unit is added. This amounts to a scale-up problem as the training of the whole enlarged network has to be performed during the constructive training process.

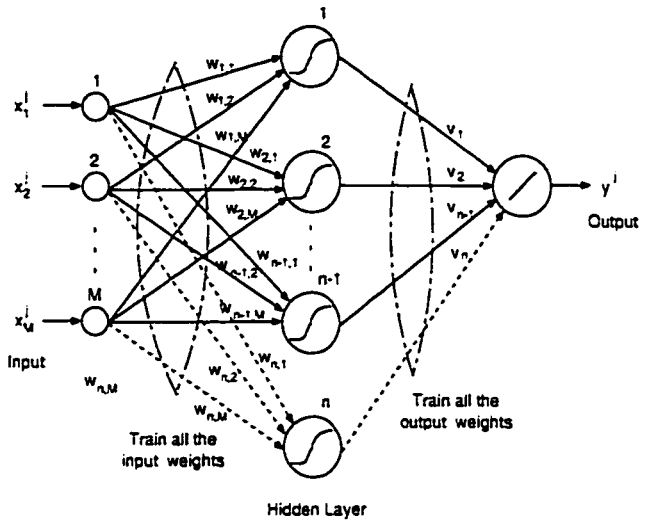


Figure 1.4: One-hidden-layer FNN under construction using DNC — the dotted lines show the newly added connections.

1.3.3 Activity-based structure level adaptation (ASLA)

A: Neuron generation

The ASLA algorithm developed by Lee [40] is motivated by the same underlying intuition used in the DNC algorithm [35]. The difference between these two structures is in the way that one adds a new hidden unit to the network. In the DNC [35], a new hidden unit with initial random input-side and output-side weights is added to the network simply when the error is stabilized. On the other hand in ASLA, when the network error is stabilized, a sensitivity measure denoted by the fluctuated distortion (FD) is evaluated for each hidden unit. If it is determined that an additional hidden unit should be added to the network, a neuron or hidden unit with the highest FD, the so-called "mother" neuron, is selected to be split into two neurons. The weights of the new neuron are selected to be identical to those of its mother neuron. After the neuron generation, a training algorithm such as BP will be applied to the enlarged network and the process is repeated until the output error

remains within a prespecified desired bound. The new neuron can be considered as a copy of its mother neuron.

Following a sufficiently long training process, a neuron that is capable of representing its map will have relatively small fluctuations, while the neuron that is incapable of this will show large fluctuations. In other words, the mother neuron is the ideal candidate that requires assistance from a new neuron. It is argued that the new neuron may be expected to help its mother neuron more efficiently and timely by copying its mother's weights as its initial weights.

B: Neuron annihilation

In ASLA, a neuron can also be annihilated from the network without affecting its performance. This is made feasible under two circumstances:

C1. When the neuron is not a "functioning" element in the network, or

C2. When the neuron is a "redundant" element in the network.

The first situation is simply identified by checking the output variance (output fluctuation) of each hidden neuron. If this is almost a constant for a given neuron, then it can be removed from the network. Furthermore, in [40] a post-sigmoidal activity variation measure was introduced for monitoring the output of each neuron. The second situation is identified by simply monitoring the correlation between the hidden neurons output values in the network. If two neurons are completely correlated, then one of them can be removed from the network. In FNNs this can be done by monitoring the dependency of the input-side weight vectors of neurons in the same layer. The neuron annihilation strategy makes the network more flexible and enhances its performance and generalization capabilities.

C: Extension of ASLA

In the ASLA developed by Lee [40], only one neuron is selected as the mother neuron, and only one new neuron is allowed to be added to the network during the

learning process. Weng and Khorasani [41] extended Lee's ASLA by allowing the selection of multiple mother neurons and the creation of multiple new neurons at any time during the learning process. This extension makes ASLA more flexible and practical. They also introduced a new algorithm to improve the training speed. The modified strategy and the new training algorithm were applied to several benchmark examples and a real-life application problem.

It should be noted that ASLA also enjoys the same advantages as that of the dynamic node creation algorithm. However, it also suffers from the scale-up problem when applied to complex problems. It is expected that ASLA may have better performance capabilities when compared to the DNC, since it employs more complicated strategies for both node creation and node annihilation. However, formal justifications of these improvements are still lacking.

1.3.4 Cascade-correlation (CC)

The CC algorithm proposed by Fahlman and Lebiere [42] is an important tool for constructing FNNs with multiple hidden layers. The CC facilitates the development of powerful high-order feature detectors even with the use of simple hidden units.

A: Cascade structure

The CC is characterized by its cascade architecture (see Figure 1.5). The algorithm starts with no hidden units, every input is connected to every output unit by a connection with an adjustable weight. New hidden units are added one at a time as required (when the training error is not small enough or when the error gets stabilized). A new hidden unit has input-side connections with not only the original network inputs but also with the output of each already existing hidden units in the current network. It has output-side connections to the original network outputs. Using this procedure, a "deep" (a FNN with many hidden layers) cascade structure will be constructed.

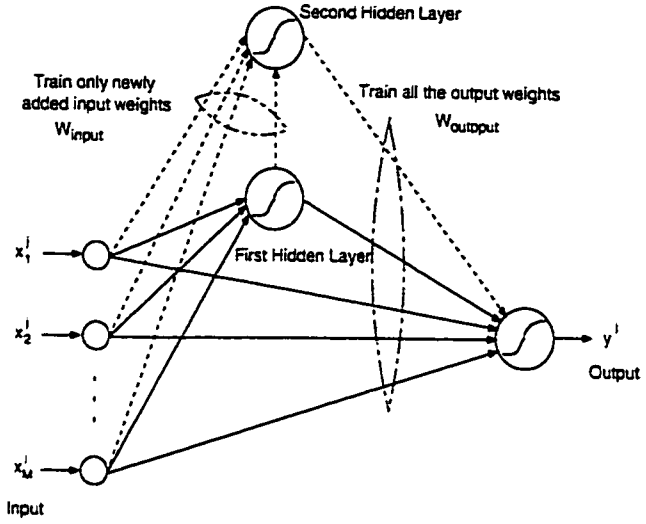


Figure 1.5: The Cascade-Correlation (CC) structure — the dotted lines show the newly added weights.

There are three major advantages with the CC algorithm:

A1. Unlike the DNC and the ASLA, CC is not concerned with the problem of initial network size, whose determination can be problem-dependent.

A2. Powerful high-order feature detectors can be constructed with simple hidden units.

A3. As will be shown below, the training of a new hidden unit is divided into input-side training and output-side training, which is then performed sequentially, resulting in computationally efficient procedure.

B: Training

In CC, there are two separate steps (or phases) for training a new hidden unit: *input-side training* and *output-side training*. Since the initial weights of a hidden unit have, in general, considerable influence on the network performance, a pool of

neuron candidates with randomly generated initial weights may be trained during the input-side training. One candidate that yields the "best" result for the objective function under consideration is incorporated into the active network, and its input-side weights will be fixed in the constructive learning process that follows. Once a new candidate is added to the network, the output-side training is then performed.

An objective function that was proposed by Fahlman and Lebiere [42] for input-side training is the maximum correlation criterion. Suppose there are $n-1$ hidden units in the network, and the error signal of the network is denoted by $e^j_{n-1,o}$, $j = 1, \cdots, P$, $o = 1, \cdots, N$, where P is the number of training samples and N is the number of outputs. Consider a candidate for the n-th hidden unit to be added to the network that is being trained. Its output is denoted by f^j_n, $j = 1, \cdots, P$. The correlation-based objective function is then given by

$$J_{input} = \sum_{o=1}^{N} \left| \sum_{j=1}^{P} (e^j_{n-1,o} - \bar{e}_{n-1,o})(f^j_n - \bar{f}_n) \right|, \tag{1.5}$$

where

$$\bar{e}_{n-1,o} = \frac{1}{P} \sum_{j=1}^{P} e^j_{n-1,o}, \tag{1.6}$$

$$\bar{f}_n = \frac{1}{P} \sum_{j=1}^{P} f^j_n. \tag{1.7}$$

The candidate that yields the maximum correlation is added to the active network, which is then followed by the output-side training in the next stage. The output-side training with n hidden units is performed according to the following least squares (LS) error criterion:

$$J_{output} = \frac{1}{2} \sum_{o=1}^{N} \sum_{j=1}^{P} (y^j_{n,o} - d^j_o)^2 \tag{1.8}$$

where $y^j_{n,o}$ is the response of the o-th output, and d^j_o is the target of the o-th output.

A fast and computationally efficient algorithm called *quickprop* was proposed by Fahlman and Lebiere [42] to maximize J_{input} and minimize J_{output}. Note that

an LS solution to the output weights can actually be obtained by simply using the pseudo-inverse operation if the output layer is linear [34, 45].

Before closing this subsection, note that the CC algorithm has also its own drawbacks. Its two major limitations are as follows:

D1. The deeper the structure is, that is the more hidden layers the network grows, 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.

D2. As the network gets deeper, the propagation delays will become longer and the fan-in of the hidden units will become complicated, making a very large scale integration (VLSI) circuitry for implementation purposes not feasible.

Towards these end, regularization-based techniques and other methods that limit the fan-in of the hidden units have been proposed in the literature to alleviate these problems [34].

1.3.5 Constructive learning of a one-hidden-layer FNN

We are now in a position to introduce some contributions to the construction of a OHL-FNN [45, 46]. In DNC algorithms, the retraining of the whole network is found to significantly increase the computation load as the network becomes larger. This begs the question: why not just train the "new" part of the network? This idea is actually realized in some sense in CC through the separation of input training for the input-side of the new hidden unit and output training for the output layer of the network. However, the fan-in problem in CC may become prohibitive as the network becomes deeper, resulting in degradation of the generalization performance. Therefore, one way to overcome the above-mentioned problems with the DNC algorithms and the CC is to construct a OHL-FNN by using the training strategy adopted in

CC (see Figure 1.6). Although constructive OHL-FNN has some advantages over CC and DNC algorithms, it also has its own problems as discussed in the Section 1.4.

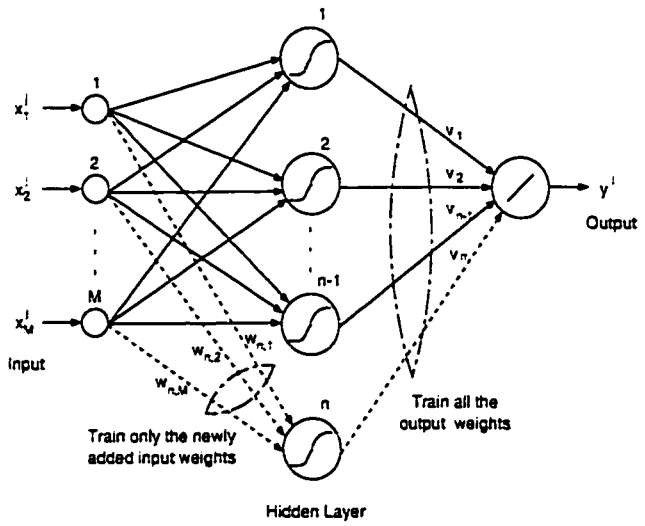


Figure 1.6: OHL-FNN under construction — the dotted lines show the newly added connections.

In [45], Kwok and Yeung investigated and compared a number of objective functions or performance indices or criteria for training new hidden units for a OHL-FNN using different regression problems. The activation functions of the hidden units were simply a sigmoidal function. These objective functions and criteria can also be directly used for training hidden units in a CC architecture. In [46], Kwok and Yeung introduced the Bayesian regularization technique into constructive OHL networks to improve their generalization performance. In this technique, a full Bayesian approach is used to accomplish the regularization by using appropriate priors regarding the network weights to favor small network weights. The regularization parameters are automatically determined by a multi-level inference process so that the user will not be required to formally provide them. More details

regarding this approach can be found in [46].

1.4 Problems with the constructive FNNs

This thesis will concentrate on the constructive learning algorithms for a OHL-FNN due to the following reasons:

R1. The OHL-FNN is simple and elegant in structure. The fan-in problem with the CC architecture is not present in this situation.

R2. OHL-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 [45].

R3. The constructive learning process is simple enough to facilitate the investigation of the training efficiency and development of other improved constructive strategies.

As can be found in [45, 46], many objective functions have been tested for several regression problems in constructing OHL-FNNs, and regularization technique was determined to be effective in improving the generalization capabilities of the constructed networks. However, analyzing the relationship between an objective function and the network performance is a very challenging problem. Towards this end, one needs to address the following questions: how "good" is an objective function? which objective function works "best"? and how does the constructive process converge? The three major issues for the constructive learning of a OHL-FNN are stated as follows:

I1. The training efficiency which is used to measure the "goodness" of an objective function in terms of the performance of the constructed FNNs, and it has been found to be quite problem-dependent [45]. Towards this end, the objective

function and the learning strategy need to be further studied. These are the motivations behind the work presented in Chapter 2.

I2. Although, it is possible to train neuron candidates with different types of activation functions, usually the same activation function is used to reduce the training time in the search space. It may be advantageous to find different activation functions for improving the training efficiency. This idea is further investigated in Chapter 4.

I3. Theoretical investigation on convergence of the training algorithm, the error dynamics, and the generalization performance with respect to the number of hidden units are other important issues that need to be investigated.

As stated before the constructive multi-hidden-layer FNNs are also considered in this thesis, since they may provide better approximation and representation capabilities to many complicated problems when compared to the OHL-FNNs. Several algorithms have been developed in the literature, such as the CC, stack learning algorithm [47], and adding-and-deleting algorithm [48].

The major limitations of CC were pointed out previously. The stack learning algorithm begins with a minimal structure consisting of the input units and the output units only, similar to the initial network that the CC starts from. 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. Clearly, this algorithm generates network that has similar structure to 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 multilayered FNN but is

computationally very intensive due to its lengthy addition-and-deletion process and the use of the BP-based training algorithm.

1.5 Overview of the thesis research

This thesis is mainly concerned with developing constructive learning algorithms for FNNs that are applied in regression (chapters 2, 3, and 4), image compression (chapter 5), and pattern recognition problems (chapters 4, and 5). It consists of the following chapters where new constructive algorithms for the FNNs at different levels of adaptation will be proposed and applied to many important practical problems.

Chapter 2 presents modifications developed for a constructive OHL-FNN through structure level adaptation where new hidden units are added one at a time when needed. These modifications attempt to improve the generalization performance and to reduce the size of the constructed OHL-FNN.

In Chapter 3, a new strategy is introduced for constructing a multi-hidden-layer FNN. This scheme is also developed using the structure level adaptation 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.

In Chapter 4, a new OHL network is introduced. In this network, each hidden unit uses a polynomial function for its activation function that is different from other units. Here, both structure level adaptation and function level adaptation are utilized in constructing the network. The function level adaptation ensures that the growing network has different activation functions such that the network may be able to capture the underlying function more efficiently. The polynomials considered consist of a series of orthonormal polynomial functions. It is shown through extensive simulations that the proposed networks constructed yield better performance when

compared to those networks with sigmoidal activation functions.

Chapter 5 is focused on the application of the constructive OHL-FNN modified in Chapter 2 to image compression and facial expression recognition. First, compression of both still images and moving frames (video sequence shots) are performed to test the generalization capability of the constructive network. The influence of quantization effects as well as comparison with the baseline JPEG scheme, and noise robustness issues are also investigated. Next, the constructive OHL-FNN is applied to facial expression recognition problem where very promising results are obtained.

Future research topics for further investigation and conclusions are provided in Chapter 6.

1.6 Conclusions

A formulation of the constructive FNN training was given. Several well-known constructive algorithms were also reviewed. The DNC, the ASLA, the CC, and the constructive OHL-FNN were all discussed. Their advantages and drawbacks were also pointed out. At the same time, we have indicated our preliminary efforts in addressing partially these drawbacks in the chapters that follow.

Chapter 2

New strategies for constructive one-hidden-layer FNNs — error scaling and input-side pruning

2.1 Introduction

Regression problem is a very important application area for neural networks (NNs). Among a large number of NN architectures, the feedforward NN (FNN) structure is one of the most widely used structures. Although OHL-FNNs have simple structures, they possess interesting representational and learning capabilities. In this chapter, we are interested particularly in incremental constructive training of OHL-FNNs.

In the incremental constructive training schemes for a OHL-FNN, input-side training and output-side training may be separated in order to reduce the training time. With the output layer activation functions taken as linear, (a situation that applies to regression applications), the output-side training can be achieved by simply computing a pseudo-inverse in the least square (LS) sense [45]. This will practically leave one with no possibilities for performance improvements. One

is then left with input-side training as a means of constructing more suitable networks that have improved generalization capabilities. The choice of a given objective function also plays a key and important role in the input-side training. There are several objective functions [45] that can be used for training the hidden units. Their efficiency, however, is found to be problem-dependent. Among the objective functions considered in [45], the correlation-based functions proposed by Fahlman and Lebiere [42] appear to be quite general and work satisfactorily for most of the regression problems. Furthermore, they have the lowest computational load among the objective functions investigated in [45].

Generally, a multivariate model-free regression problem can be described as follows. Suppose one is given P pairs of vectors

$$(\mathbf{d}^j, \mathbf{x}^j) = (d_1^j, d_2^j, \cdots, d_N^j; x_1^j, x_2^j, \cdots, x_M^j), \quad j = 1, 2, \cdots, P \tag{2.1}$$

that are generated from unknown models

$$d_i^j = g_i(\mathbf{x}^j) + \epsilon_i^j, \quad i = 1, 2, \cdots, N, \quad j = 1, 2, \cdots, P \tag{2.2}$$

where the $\{\mathbf{d}^j\}$'s are called the multivariate "response" vectors and $\{\mathbf{x}^j\}$'s are called the "independent variables" or the "carriers", and M and N are dimensions of $\mathbf{x}$ and $\mathbf{d}$, respectively. The $\{g_i\}$'s are unknown smooth non-parametric or model-free functions from M-dimensional Euclidean space to the real line:

$$g_i : R^M \longrightarrow R, \quad i = 1, 2, \cdots, N. \tag{2.3}$$

The $\{\epsilon_i^j\}$'s are random variables (or noise) with zero mean $E[\epsilon_i^j] = 0$, and are independent of $\{\mathbf{x}^j\}$. Usually, $\{\epsilon_i^j\}$'s are assumed to be independent and identically distributed (*iid*). The goal of regression is to obtain estimators, $\hat{g}_1, \hat{g}_2, \cdots, \hat{g}_N$ that are functions of the data $(\mathbf{d}^j, \mathbf{x}^j)$, $j = 1, 2, \cdots, P$, to best approximate the unknown functions, $g_1, g_2, \cdots, g_N$, and use these estimators to predict (generalize) a new $\mathbf{d}$

given a new $\mathbf{x}$:

$$\hat{d}_i = \hat{g}_i(\mathbf{x}), \quad i = 1, 2, \cdots, N. \tag{2.4}$$

Suppose here that the regression problem has an M-dimensional input vector and a one-dimensional output scalar ($N = 1$). The j-th input and the output patterns are denoted by $\mathbf{x}^j = (x_0^j, x_1^j, x_2^j, \cdots, x_M^j)$ and d^j (target), respectively, where $j = 1, 2, ..., P$, and $x_0^j = 1$ for all j as representing the bias. The constructive OHL algorithm [45] may start from a small network, say a one hidden unit network, adapted by a backpropagation type learning algorithm. At any given point during the constructive training process, say there are $n - 1$ hidden units in the hidden layer, and the problem then is to train the n-th hidden unit that is to be added to the network. All the hidden units have identical sigmoidal activation function. The network is depicted in Figure 2.1. A candidate that maximizes the correlation-based objective function will be selected from a pool of candidates, and is then incorporated into the network as an n-th hidden unit (for details refer to next section). The input to the n-th hidden unit is given by

$$s_n^j = \sum_{i=0}^{M} w_{n,i} x_i^j, \quad j = 1, 2, \cdots, P \tag{2.5}$$

where $w_{n,i}$ is the weight from the i-th input to the n-th hidden unit, $w_{n,0}$ is the so-called bias of the n-th hidden unit and its input is set to $x_0^j = 1$. Before proceeding to the output-side of this new hidden unit, one has to first train its input-side weight(s). This is the so-called input-side training phase. Once the input-side training is accomplished, the output of the hidden unit can then be expressed as

$$f_n(s_n^j) = f(\sum_{i=0}^{M} w_{n,i} x_i^j), \quad j = 1, 2, \cdots, P \tag{2.6}$$

where $f(\cdot)$ is the sigmoidal activation function of the hidden unit defined as an example according to $f(x) = (1 + e^{-\lambda x})^{-1}$, where λ is the slope parameter of the sigmoid. The network output with n hidden units may now be expressed as follows:

$$y_n^j = \sum_{k=0}^{n} v_k f_k(s_k^j), \quad j = 1, 2, \cdots, P \tag{2.7}$$

where v_k, $(k = 1, 2, \cdots, n)$ are output-side weights of the k-th hidden unit, and v_0 is the bias of the output unit with its input being fixed to $f_0 = 1$. The corresponding output error is now given by

$$\begin{aligned} e_n^j &= d^j - y_n^j \\ &= d^j - \sum_{k=0}^{n} v_k f_k(s_k^j), \quad j = 1, 2, \cdots, P. \end{aligned} \tag{2.8}$$

Subsequently, the output-side training is performed by solving the following LS problem given that the output layer is linear,

$$\begin{aligned} J_{output} &= \frac{1}{2}\sum_{j=1}^{P}(e_n^j)^2 \\ &= \frac{1}{2}\sum_{j=1}^{P}(d^j - y_n^j)^2 \\ &= \frac{1}{2}\sum_{j=1}^{P}\left\{d^j - \sum_{k=0}^{n} v_k f_k(s_k^j)\right\}^2. \end{aligned} \tag{2.9}$$

Following the output-side training, a new error signal e_n^j is calculated from (2.8) for the next cycle of input-side training associated with the $(n+1)$-th hidden unit and its corresponding output-side training.

The remainder of this chapter is organized as follows. Section 2.2 presents the concepts of the error scaling that is introduced to improve the performance of the constructive training procedure. In Section 2.3, two pruning strategies are proposed to remove (delete), during the constructive training process, the input-side connections that have "minimal" contributions to the input-side training. Since these pruning techniques are used locally, the generalization performance of the resulting networks may not be significantly improved, however networks with fewer connections will be constructed that should be more advantageous for implementation purposes. In Section 2.4, convergence of the proposed constructive algorithm will be discussed, and an optimal objective function for the input-side training will be obtained that is different from all the other objective functions given in [45]. This

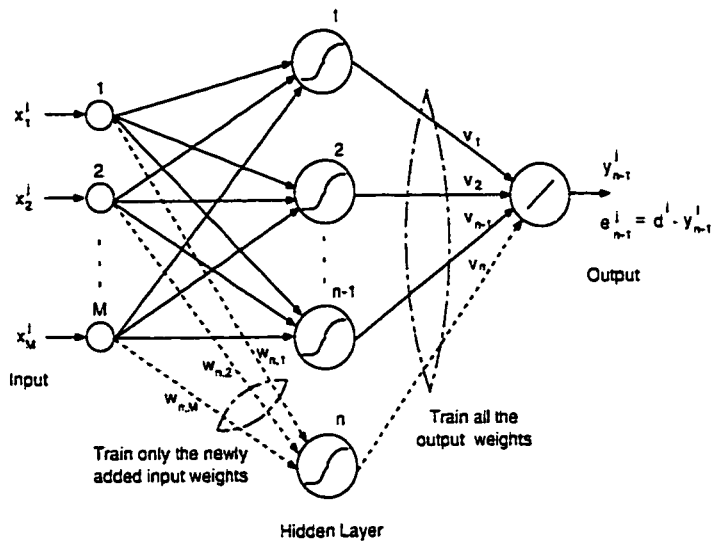


Figure 2.1: Structure of a constructive OHL-FNN.

objective function is theoretically expected to outperform the objective functions investigated in [45]. Section 2.5 concludes this chapter.

2.2 Error scaling strategy for input-side training

In this section, the efficiency of the correlation-based objective function is investigated. Without loss of any generality, a regression problem with only one output is considered. The correlation-based objective function in this case is given as follows [42]:

$$J_{input} = \left| \sum_{j=1}^{P} (e_{n-1}^{j} - \bar{e}_{n-1})(f_n(s_n^j) - \bar{f}_n) \right| \tag{2.10}$$

where

$$\bar{e}_{n-1} = \frac{1}{P} \sum_{j=1}^{P} e_{n-1}^{j} \tag{2.11}$$

$$\bar{f}_n = \frac{1}{P}\sum_{j=1}^{P} f_n(s_n^j) \tag{2.12}$$

with $\bar{e}_{n-1}$ and $\bar{f}_n$ denoting the mean values of the training error and the output of the n-th hidden unit over the entire training samples, respectively. The above objective function attempts to maximize the correlation between the unbiased training error and the output of the new hidden unit. In other words, the output training error is more likely to be reduced by as largest as possible when the new hidden unit is added to the network. The derivative of J_{input} with respect to the weight $w_{n,i}$ is calculated as (for details refer to Appendix A for a multi-output case)

$$\frac{\partial J_{input}}{\partial w_{n,i}} = C_0 \sum_{j=1}^{P} (e_{n-1}^j - \bar{e}_{n-1}) f_n'(s_n^j) x_i^j. \tag{2.13}$$

where

$$C_0 = sgn(\sum_{j=1}^{P} (e_{n-1}^j - \bar{e}_{n-1})(f_n(s_n^j) - \bar{f}_n)). \tag{2.14}$$

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. The explicit weight adjustment rules are also given in detail in Appendix A for an output layer with multiple neurons. Note that the above expressions may be obtained by simply deleting subscript "o" from (A.5)-(A.6). From (2.10) and (2.13), the following observations can be stated:

O1. Maximization of the objective function (2.10) requires that the output of the new hidden unit $f_n(s_n^j)$ bear resemblance to the error signal e_{n-1}^j as close as possible. The optimal situation may be expressed as

$$f_n(s_n^j) = f(s_n^j) = \lambda e_{n-1}^j \tag{2.15}$$

for all the training patterns, where λ is a sufficiently large constant. Under this condition, it can be shown that the addition of the $n-$th hidden unit will make the network output error zero, which implies that a "perfect" training may be achieved. However, in most regression problems this ideal situation

is impossible to achieve and therefore, one is then required to perform the input-side training to approach the above ideal case. Refer to Section 2.4 for further details on the convergence of a constructive training algorithm.

O2. Since $f(s_n^j)$ is upper and lower bounded (saturated for large or small values of s_n^j) to the interval $[0, 1]$ as a result of the nature of the sigmoidal activation function, (2.15) will not be satisfied if certain errors e_{n-1}^j for some training samples reside outside the ranges of $f(\cdot)$. In this case, the activation function will be forced into its saturation regions, in order to achieve as much "resemblance" as possible to the corresponding error signal. The derivative of the activation function around the saturation region will be zero, resulting in no contribution to (2.10). Equivalently, the training samples that have resulted this undesirable situation will consequently have no contribution to the input-side training.

O3. Based on the above observations one can envisage the following simple remedy: why not try tc position the undesirable error signal samples into the operational range of the activation functions (where they have non-zero derivatives) during the input-side training? This is the motivation behind the proposed "error scaling technique" that will be developed below.

2.2.1 Error scaling

During the input-side training phase, if the error signal e_{n-1}^j varies within the range $[e_{n-1,min}, e_{n-1,max}]$ that is beyond the active range $[f_{min}, f_{max}]$ of the activation function of a hidden unit, then the particular hidden unit can not follow these errors that are outside its active range. Consequently, the error signal range $[e_{n-1,min}, e_{n-1,max}]$ should ideally be mapped into the active range of the hidden unit $[f_{min}, f_{max}]$, by invoking a linear coordinate transformation, as follows:

$$f_{max} = C_1 e_{n-1,max} + C_2 \tag{2.16}$$

$$f_{min} = C_1 e_{n-1,min} + C_2. \tag{2.17}$$

Note that only linear transformations can be utilized here if one wants to preserve the waveform of the error signal. Solving the above expressions for the parameters C_1 and C_2 yields

$$C_1 = \frac{f_{max} - f_{min}}{e_{n-1,max} - e_{n-1,min}}, \tag{2.18}$$

$$C_2 = \frac{e_{n-1,max} f_{min} - e_{n-1,min} f_{max}}{e_{n-1,max} - e_{n-1,min}}. \tag{2.19}$$

Therefore, the error signal e^j_{n-1} is linearly transformed into $e^j_{n-1,new}$ according to

$$e^j_{n-1,new} = \begin{cases} C_1 e^j_{n-1} + C_2 & C_1 < 1 \\ e^j_{n-1} - \bar{e}_{n-1} + \frac{f_{max}+f_{min}}{2} & C_1 \geq 1 \end{cases} \tag{2.20}$$

The error signal e^j_{n-1} in (2.10) will now be replaced by $e^j_{n-1,new}$ which is calculated from the above equation. If the activation function of the hidden units is a log-sigmoidal function, then its active range, for example, may be set to [0.1, 0.9]. It is clear that this active range is confined within the upper and lower bounds of the activation function. By narrowing the bounds the active range can yield more effective training. During simulations for this chapter to determine the active range for a log-sigmoidal function, it has been found that the active range [0.1, 0.9] yielded good training and generalization results. It is not difficult to determine from the above equation that if $e_{n-1,max}$ and $e_{n-1,min}$ are chosen such that the mean value of the error signal $\bar{e}_{n-1}$ is $\frac{1}{2}(e_{n-1,max} + e_{n-1,min})$, one gets

$$\bar{e}_{n-1,new} = \frac{f_{max} + f_{min}}{2} \tag{2.21}$$

regardless of C_1. This is a point where the activation function reaches its maximum first-order derivative. The replacement of e^j_{n-1} by $e^j_{n-1,new}$ is now expected to improve the efficiency of the input-side training.

What is now required to be defined are the bounds for the error signal e_{n-1}^j. Since the correlation between $f_n(s_n^j)$ and e_{n-1}^j is statistically defined, the bounds for e_{n-1}^j have to also be selected in a statistical sense rather than a simple fixed values. Consequently, following the above reasoning, we take the following values as the bounds of e_{n-1}^j, i.e.,

$$e_{n-1,max} = \bar{e}_{n-1} + 3\sigma_{e_{n-1}}, \tag{2.22}$$

$$e_{n-1,min} = \bar{e}_{n-1} - 3\sigma_{e_{n-1}} \tag{2.23}$$

where $\sigma_{e_{n-1}}$ is the standard deviation of e_{n-1}^j.

The well-known "quickprop" algorithm is used to maximize the correlation criterion (2.10) [42, 45]. This proposed technique will be applied to numerous other examples to confirm and demonstrate its effectiveness.

2.2.2 Simulation results

In this subsection, several examples are worked out to some details to demonstrate the effectiveness of the proposed error scaling technique. The examples that follow are regression functions that were used in many previous work [45] (and references therein). In all the following examples the performance of a network is measured by the fraction of variance unexplained (FVU) [45] which is defined as

$$FVU = \frac{\sum_{j=1}^{P}(\hat{g}(x^j) - g(x^j))^2}{\sum_{j=1}^{P}(g(x^j) - \bar{g})^2} \tag{2.24}$$

$$\bar{g} = \frac{1}{P}\sum_{j=1}^{P} g(x^j)$$

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

$$\begin{aligned} SNR &= 10\ log_{10}\left\{\frac{\sum_{j=1}^{P}(g(x^j)-\bar{g})^2}{\sum_{j=1}^{P}(\epsilon^j-\bar{\epsilon})^2}\right\} \\ \bar{\epsilon} &= \frac{1}{P}\sum_{j=1}^{P}\epsilon^j \end{aligned} \tag{2.25}$$

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

Example : Let us consider the following five (5) regression functions that were studied in [45]:

(a) *Complicated interaction function* (CIF)

$$g(x_1,x_2) = 1.9\left(1.35 + e^{x_1}e^{-x_2}\sin(13(x_1-0.6)^2)\sin(7x_2)\right). \tag{2.26}$$

(b) *Additive function* (AF)

$$\begin{aligned} g(x_1,x_2) &= 1.3356\ (\ 1.5(1-x_1) + e^{2x_1-1}\sin(3\pi(x_1-0.6)^2) \\ &\quad +e^{3(x_2-0.5)}\sin(4\pi(x_2-0.9)^2)\ ) \end{aligned} \tag{2.27}$$

(c) *Harmonic function* (HF)

$$\begin{aligned} g(x_1,x_2) &= 42.659\ (\ 0.1 + (x_1-0.5)(0.05 + (x_1-0.5)^4 \\ &\quad -10(x_1-0.5)^2(x_2-0.5)^2 + 5(x_2-0.5)^4)\ ) \end{aligned} \tag{2.28}$$

(d) *Radial function* (RF)

$$g(x_1, x_2) = 24.234 \left((x_1 - 0.5)^2 + (x_2 - 0.5)^2 \right) \times \left(0.75 - (x_1 - 0.5)^2 - (x_2 - 0.5)^2 \right) \quad (2.29)$$

(e) *Simple interaction function* (SIF)

$$g(x_1, x_2) = 10.391 \left((x_1 - 0.4)(x_2 - 0.6) + 0.36 \right) \quad (2.30)$$

The surfaces of these functions are shown in Figures 2.2(a)-2.6(a).

For each function two hundred and twenty-five (225) uniformly distributed random points were generated from an interval [0,1] for each input dimension for network training. Ten thousands (10000) uniformly sampled points from the same interval without additive noise were used to test the generalization performance of a trained network with different number of hidden units. Forty (40) independent runs were performed for an ensemble-averaged evaluation for two cases where SNRs are 10 [dB] and 0 [dB], respectively. Mean FVUs values for the training and the generalization performance of networks trained with and without error scaling for the above regression functions are summarized in Table 2.1. Selected generalized surfaces of the above functions by the networks constructed using samples with 10 [dB] SNR are also provided for comparison in Figures 2.2(b),(c),(d) - 2.6(b),(c),(d).

Comparisons between the training and the generalization performances of the previous (without error scaling) and the new (with error scaling) algorithms for all the five functions are depicted in Figures 2.7-2.11, where the SNR is 0 [dB]. From Table 2.1 and Figures 2.7-2.11, the following comments are in order:

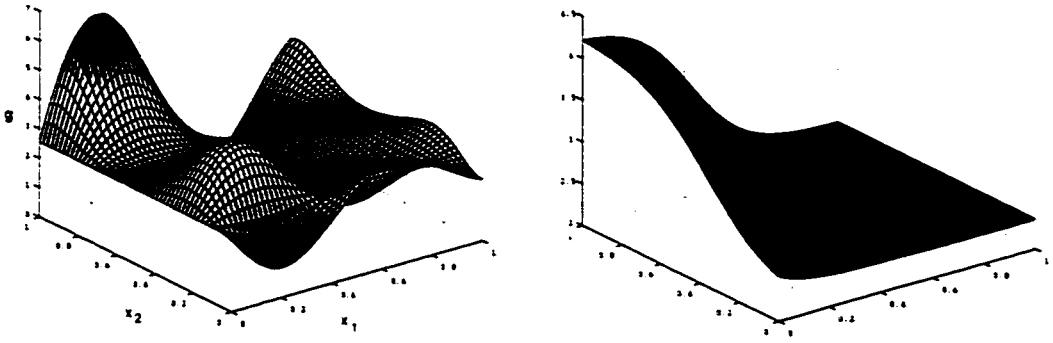
(1) The proposed training technique provides improved generalization performance compared to the standard constructive FNNs, especially for large noise scenarios.

(2) The new error scaling technique demonstrates a capability for avoiding over-training in all the regression functions considered when compared with the previous technique. Specifically, even though the proposed technique yields larger training FVUs than the previous technique as the network grows, it generates improved generalization FVUs compared to previous methods.

(3) In some cases the new technique can not completely prevent over-training as the generalization FVUs also increase slightly as more hidden units are added to the net.

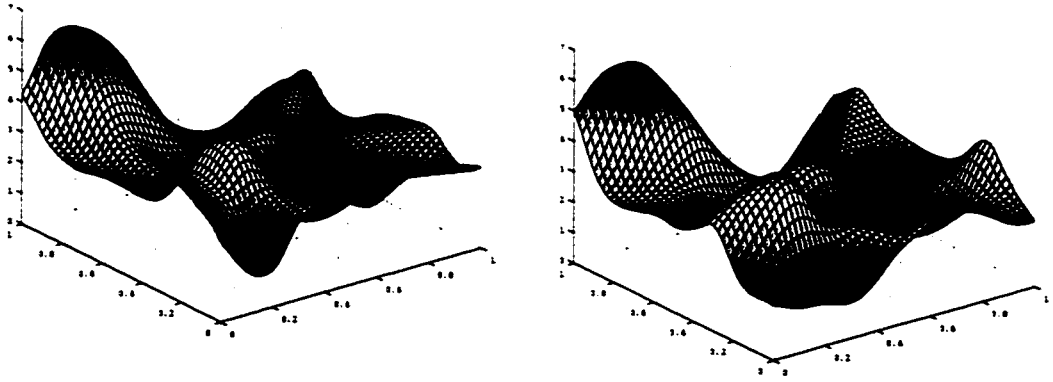
Table 2.1: Mean FVUs values of the training and the generalization performance of networks constructed with ("new") and without ("previous") error scaling technique as a function of the number of hidden units.

			Number of hidden units							
			Training				Generalization			
Fun.	SNR [dB]	Approach	2	5	10	20	2	5	10	20
CIF	10	previous	0.659	0.429	0.181	0.089	0.685	0.452	0.214	0.108
		new	0.661	0.409	0.164	0.095	0.686	0.433	0.187	0.088
	0	previous	0.778	0.639	0.501	0.404	0.646	0.548	0.362	0.235
		new	0.779	0.643	0.501	0.454	0.646	0.512	0.290	0.200
AF	10	previous	0.697	0.207	0.126	0.092	0.723	0.171	0.094	0.048
		new	0.634	0.182	0.125	0.095	0.644	0.135	0.079	0.040
	0	previous	0.791	0.500	0.406	0.350	0.699	0.270	0.225	0.313
		new	0.797	0.494	0.426	0.406	0.665	0.184	0.155	0.190
HF	10	previous	0.770	0.557	0.367	0.171	0.888	0.678	0.499	0.290
		new	0.777	0.551	0.336	0.157	0.894	0.657	0.421	0.219
	0	previous	0.882	0.712	0.573	0.407	0.898	0.783	0.565	0.449
		new	0.885	0.729	0.565	0.461	0.898	0.838	0.493	0.332
RF	10	previous	0.625	0.240	0.125	0.072	0.691	0.243	0.098	0.036
		new	0.627	0.200	0.110	0.073	0.684	0.191	0.065	0.024
	0	previous	0.658	0.493	0.417	0.356	0.632	0.267	0.258	0.405
		new	0.658	0.498	0.441	0.415	0.622	0.223	0.185	0.264
SIF	10	previous	0.275	0.136	0.098	0.082	0.281	0.101	0.054	0.032
		new	0.275	0.132	0.097	0.085	0.276	0.085	0.047	0.022
	0	previous	0.634	0.454	0.400	0.361	0.357	0.182	0.164	0.206
		new	0.628	0.453	0.419	0.406	0.344	0.174	0.123	0.154



(a) CIF (original)

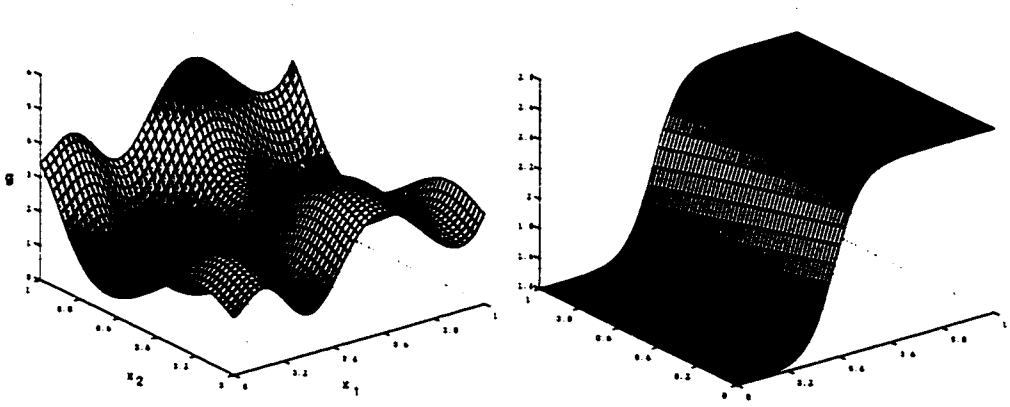
(b) Generalized CIF with 1 hidden unit (HU)



(c) Generalized CIF with 10 HUs

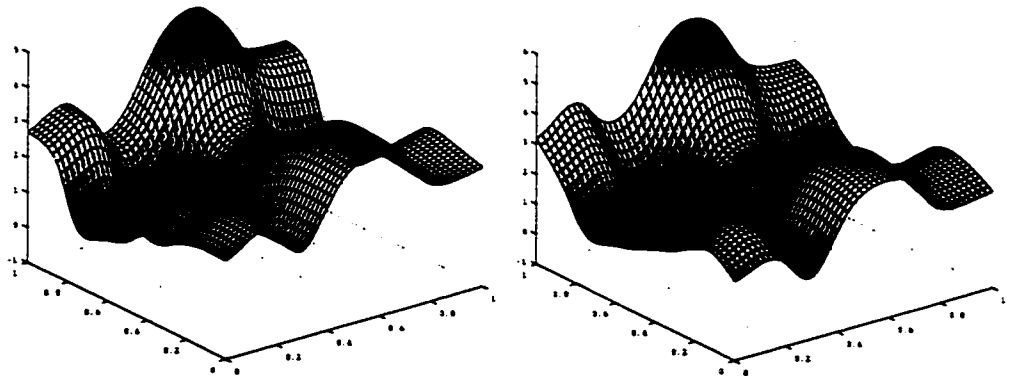
(d) Generalized CIF with 20 HUs

Figure 2.2: The original and the generalized CIF function.



(a) AF (original)

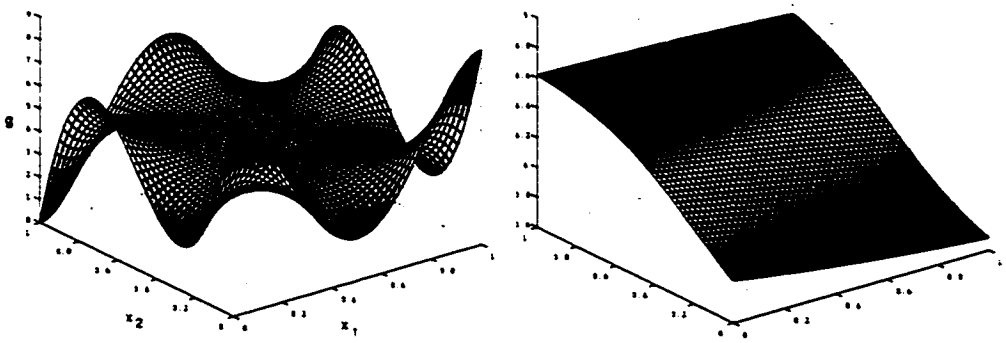
(b) Generalized AF with 1 hidden units (HU)



(c) Generalized AF with 10 HUs

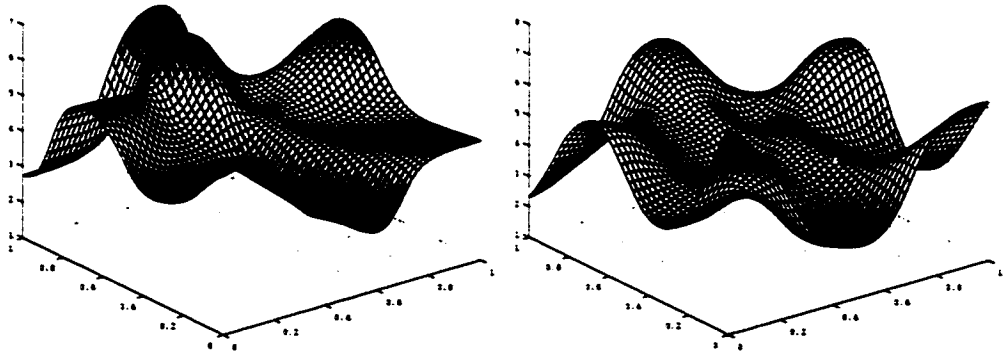
(d) Generalized AF with 20 HUs

Figure 2.3: The original and the generalized AF function.



(a) HF (original)

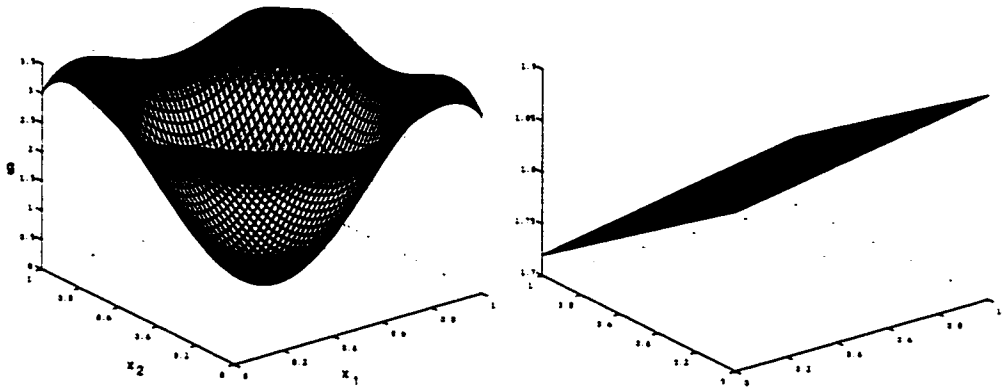
(b) Generalized HF with 1 hidden unite (HU)



(c) Generalized HF with 10 HUs

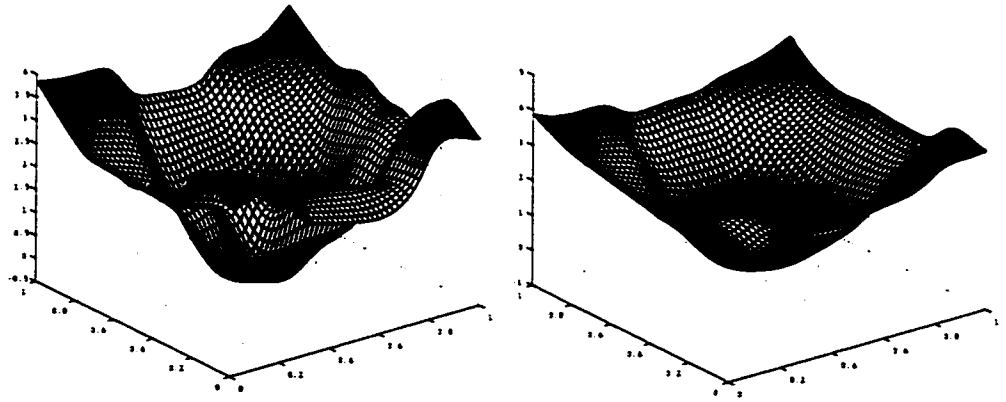
(d) Generalized HF with 20 HUs

Figure 2.4: The original and the generalized HF function.



(a) RF (original)

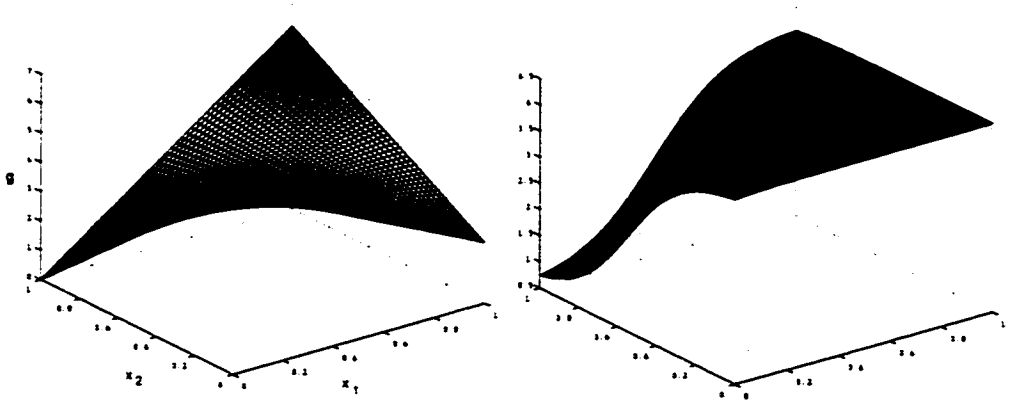
(b) Generalized RF with 1 hidden unit (HU)



(c) Generalized RF with 10 HUs

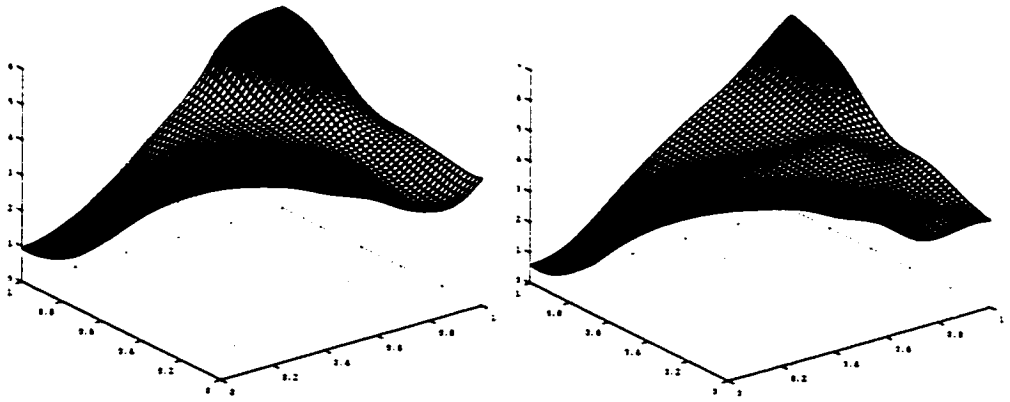
(d) Generalized RF with 20 HUs

Figure 2.5: The original and the generalized RF function.



(a) SIF (original)

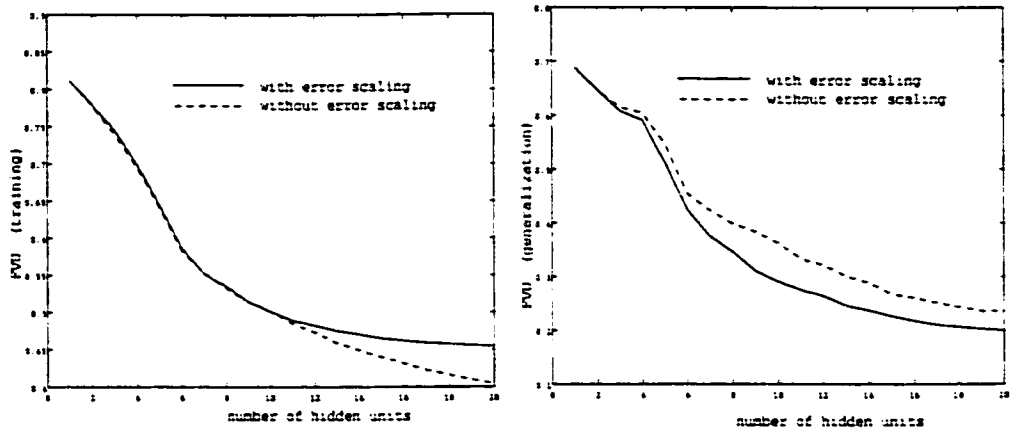
(b) Generalized SIF with 1 hidden unit (HU)



(c) Generalized SIF with 10 HUs

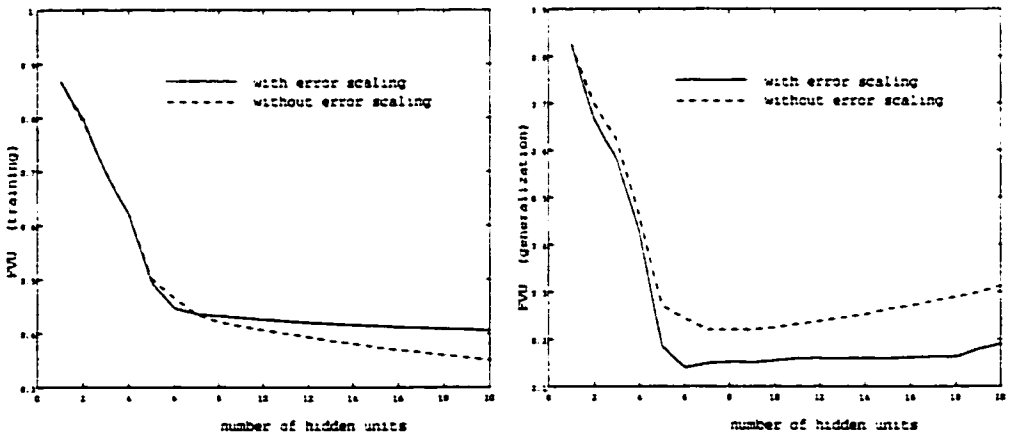
(d) Generalized SIF with 20 HUs

Figure 2.6: The original and the generalized SIF function.



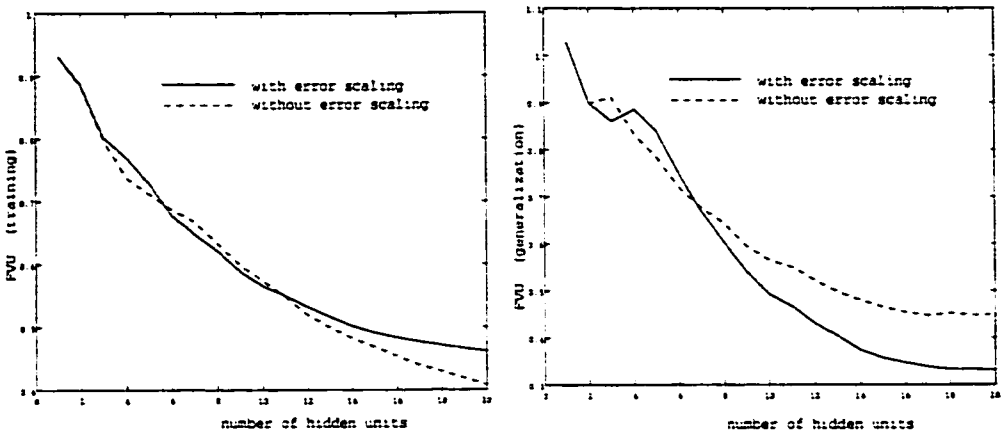
(a) FVUs for the training (b) FVUs for the generalization

Figure 2.7: Comparisons between the mean FVUs of the training and the generalization with and without error scaling for the CIF regression problem (SNR=0 [dB], 40 runs).



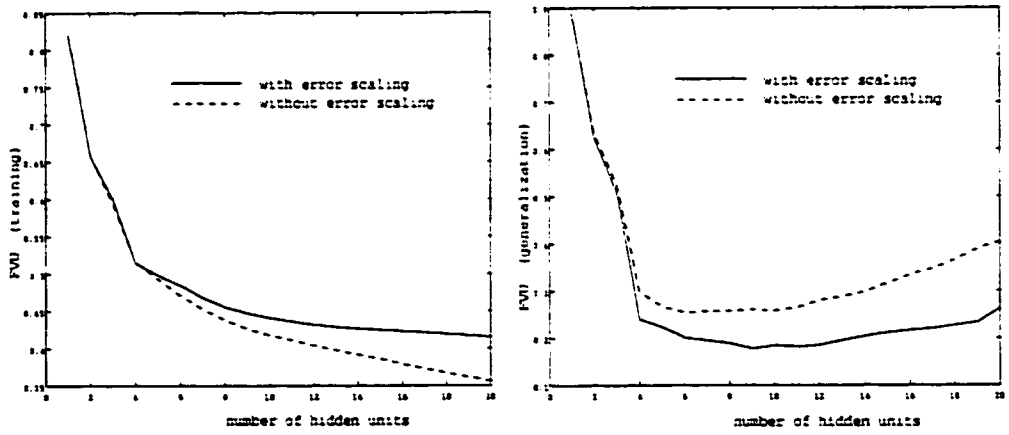
(a) FVUs for the training (b) FVUs for the generalization

Figure 2.8: Comparisons between the mean FVUs of the training and the generalization with and without error scaling for the AF regression problem (SNR=0 [dB], 40 runs).



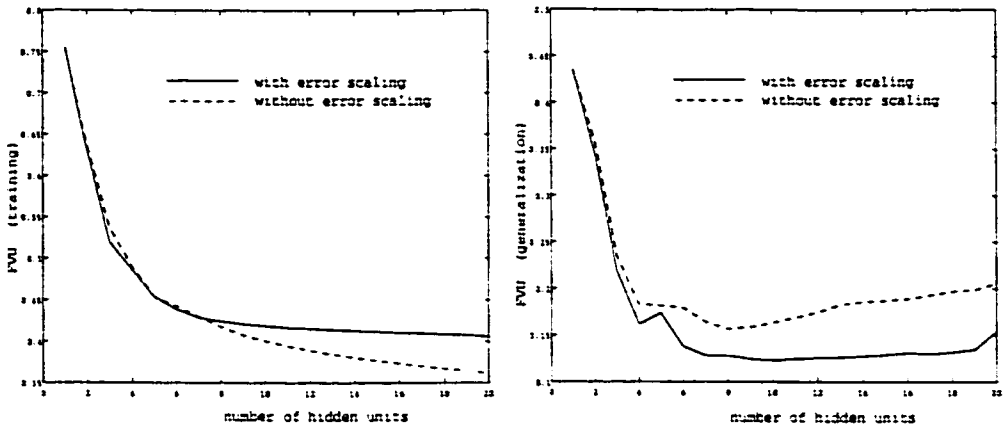
(a) FVUs for the training (b) FVUs for the generalization

Figure 2.9: Comparisons between the mean FVUs of the training and the generalization with and without error scaling for the HF regression problem (SNR=0 [dB], 40 runs).



(a) FVUs for the training (b) FVUs for the generalization

Figure 2.10: Comparisons between the mean FVUs of the training and the generalization with and without error scaling for the RF regression problem (SNR=0 [dB], 40 runs).



(a) FVUs for the training (b) FVUs for the generalization

Figure 2.11: Comparisons between the mean FVUs of the training and the generalization with and without error scaling for the SIF regression problem (SNR=0 [dB], 40 runs).

2.3 Input-side pruning strategies

In the input-side training, one can have one or a pool of candidates to train a new hidden unit. In the latter case, the neuron that results in the maximum objective function will be selected as the best candidate. This candidate is incorporated into the network and its input-side weights are frozen in the subsequent training process that follows. However, certain input-side weights may contribute little, if any, to the objective function or indirectly to the reduction of the training error. These connections should first be detected and then removed through a pruning technique. Pruning these connections is expected to produce a smaller network without compromising the performance of the network. Note that the pruning operation is carried out "locally", and therefore the generalization performance of the final network will not be improved significantly, since the general pruning-and-backfitting performed in standard fixed size network pruning is not implemented here. In this

thesis we propose two types of sensitivity functions for the purpose of formalizing the input-side weight pruning process.

2.3.1 Sensitivity based on training error (Method A)

Let us reproduce here the squared output error criterion given in (2.9) for a single output network

$$J_{output} = \frac{1}{2}\sum_{j=1}^{P}\left(y_n^j - d^j\right)^2 \tag{2.31}$$

If J_{output} is locally minimized by incorporating the n-th hidden unit, the error surface with respect to the input-side weight(s) denoted by $\mathbf{W}_n = (w_{n,0}, w_{n,1}, \cdots, w_{n,M})$ of the new hidden unit will appear to be bowl-shaped in a very small vicinity of that local minimum. This implies that the second-order derivatives of J_{output} with respect to each weight of that unit will be positive at the local minimum. If this is not the case for a given weight, that weight would then make no contribution to reducing the training error. The first derivative of J_{output} with respect to an input-side weight $w_{n,i}$ of the n-th hidden unit is calculated as follows

$$\frac{\partial J_{output}}{\partial w_{n,i}} = v_n \sum_{j=1}^{P} x_i^j f'(s_n^j)(y_n^j - d^j) \tag{2.32}$$

The second-order derivative (sensitivity) function may now be evaluated according to

$$\begin{aligned} S_{n,i}^{A} &= \frac{\partial^2 J_{output}}{\partial w_{n,i}^2} \\ &= v_n^2 \sum_{j=1}^{P}(x_i^j)^2(f'(s_n^j))^2 + v_n \sum_{j=1}^{P}(x_i^j)^2(y_n^j - d^j)f''(s_n^j) \end{aligned} \tag{2.33}$$

The above criterion (2.33) for pruning a weight was used for pruning BP-based trained NNs in [25]. But here it is used for pruning the input-side weights only during the input-side training phase. Because the input-side training is not performed under

the above squared output error criterion, this sensitivity is not a direct criterion for the input-side pruning.

When the output-side training is completed, the second-order derivative or sensitivity function (2.33) with respect to each weight of the new hidden unit is calculated. The weights that result in negative sensitivity are set to zero or pruned. Subsequently, the output of the new hidden unit is recalculated and the output-side training is performed again. Clearly, pruning the weights that make no contribution in reducing the training error may be achieved only once, however output-side training has to be performed twice: once before and the other time after the pruning. Furthermore, weights that have relatively very "small" second-order derivatives, for example lower than some prespecified threshold (we denote it by *pruning level*), can also be removed without significantly affecting the performance of the network.

2.3.2 Sensitivity based on correlation (Method B)

Suppose that the best candidate for the n-th hidden unit to be added to the network results in an objective function $J_{max,n}$. Then, sensitivity of each weight may be defined as

$$S^B_{n,i} = J_{max,n} - J_{input}(w_{n,i} = 0), \quad i = 1, 2, \cdots, M, \tag{2.34}$$

where $J_{input}(w_{n,i} = 0)$ is the value of the objective function when $w_{n,i}$ is set to zero while other connections are unchanged. Note that the bias is usually not pruned. The above sensitivity function measures the contribution of each connection to the objective function. The largest value for the n-th hidden unit sensitivity is denoted by S^{max}_n. If $S^B_{n,i} \leq 0$, and/or is very small compared to S^{max}_n, say 3% (*pruning level*: $= (S^{max}_n - S^B_{n,i})/S^{max}_n$) of it, then the weight $w_{n,i}$ is removed. After the pruning is performed, the output of the hidden unit $f(s^j_n)$ is re-evaluated and the output-side training is performed.

It should be noted that it is very difficult to compare the above two pruning

methods analytically. However intuitively, one may say that Method B is likely to provide better performance than Method A, since the sensitivity criterion used in Method B is directly related to the cost function J_{input} used for input-side weight training.

2.3.3 Simulation results

To confirm the validity of the pruning methods A and B, we present in this section some simulation results. Intuitively, the larger the dimension of the input vector, the more effective the above pruning techniques would be. In all the following examples, ten (10) independent runs are performed to gauge an average evaluation. The SNR in all examples is set to 10 [dB].

Example 1: The first example is the 2-dimensional function CIF [45, 50] used in the previous section. Again, two hundred and twenty-five (225) uniformly distributed random points were generated from an interval [0,1] for network training. Ten thousands (10000) uniformly sampled points from the same interval without additive noise were used to test the generalization performance of the trained network. The pruning level is set to 5%. Figure 2.12 shows the generalization errors with and without pruning method A (error scaling is performed for both cases), and the cumulative number of weights pruned using method A. Figure 2.13 depicts the corresponding results using the pruning method B.

Example 2: The second example is a simple 3-dimensional analytical function (SAF) [52] given by

$$g(x_1, x_2, x_3) = \frac{1}{1 + e^{-e^{x_1} + (x_2 - 0.5)^2 + 3\sin(\pi x_3)}}. \tag{2.35}$$

One thousand (1000) random samples with additive noise were generated for network training, and five thousands (5000) random samples without additive noise were

generated for verifying the network generalization. The pruning level in this case is set to 1%. Figure 2.14 shows the generalization errors with and without pruning method A (error scaling is performed for both cases), and the cumulative number of weights pruned using method A. Similar results using the pruning method B are shown in Figure 2.15. The pruning level is again set to 1%.

From Figures 2.12 and 2.13, one may observe that pruning method A did not remove any weight while pruning method B pruned on average only one and half weights from a network that has 20 hidden units. This shows that the two proposed pruning methods do not indicate a significant advantage for problems having a lower order input dimension.

Figures 2.14 and 2.15 demonstrate that both methods have pruned some weights, but with method B showing more effectiveness than method A, due to the fact that more weights could be pruned in method B than method A, even though both methods yield practically the same FVUs for both training and generalization processes.

Example 3: The third example is a 5-dimensional function (HAS) [53] that is given by

$$\begin{aligned} g(x_1, x_2, x_3, x_4, x_5) &= 0.0647(12 + 3x_1 - 3.5x_2^2 + 7.2x_3^3) \\ &\quad \times (1 + \cos 4\pi x_4)(1 + 0.8 \sin 3\pi x_5). \end{aligned} \tag{2.36}$$

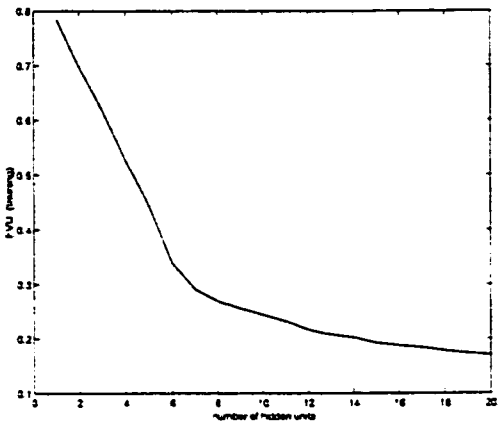
Three thousands (3000) random samples with additive noise were generated for network training, and ten thousands (10000) random samples without additive noise were generated for verifying the network generalization capabilities. The pruning level in this case is set to 0.2%. Figures 2.16 and 2.17 illustrate the results achieved using methods A and B, respectively.

For a quantitative comparison, the results obtained using method B for the above three examples are summarized in Table 2.2.

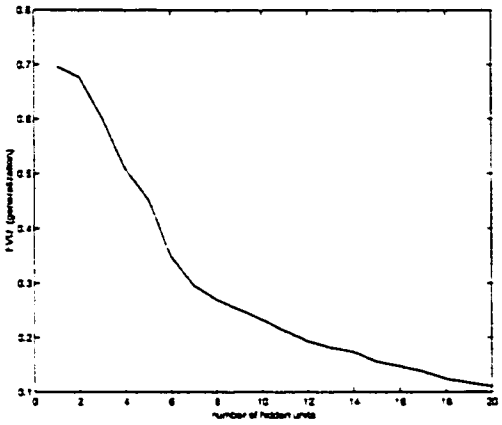
Table 2.2: Percentage (η) of the number of connections pruned by the proposed method B with respect to that of the full input-side connections (10 Runs, FVU : generalization error without pruning, FVU_p : generalization error with pruning).

		Number of hidden units								
Fun.	Pruning level (%)	5			10			15		
		FVU	FVU_p	η (%)	FVU	FVU_p	η (%)	FVU	FVU_p	η (%)
CIF	5	0.377	0.377	0.00	0.218	0.213	3.50	0.158	0.161	4.33
SAF	1	0.068	0.091	25.3	0.022	0.024	19.2	0.018	0.018	15.1
HAS	0.2	0.671	0.660	22.8	0.208	0.216	27.2	0.195	0.202	20.0

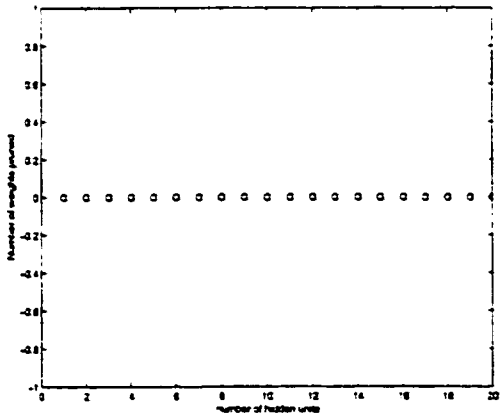
To further confirm and demonstrate the effectiveness of the proposed two pruning methods, the constructive OHL network is applied to image compression problem in Chapter 5 where the networks are pruned using the ideas developed in this chapter.



(a) FVUs for the training

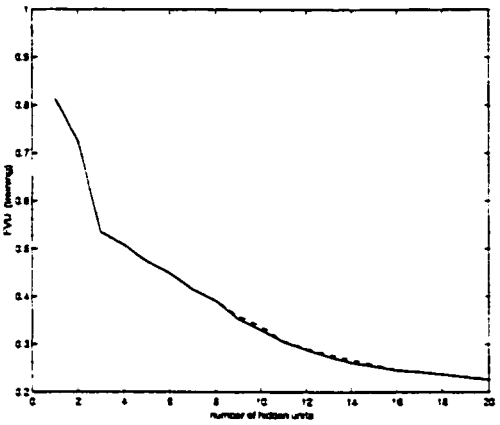


(b) FVUs for the generalization

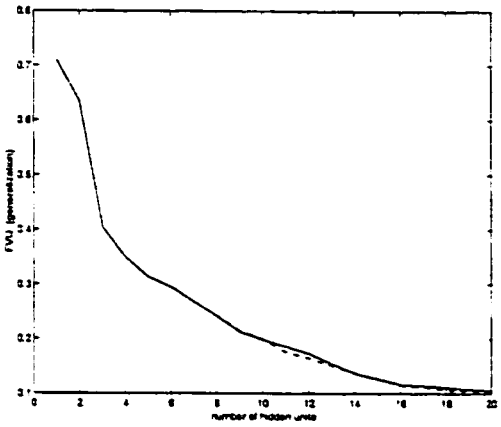


(c) Cumulative number of weights pruned

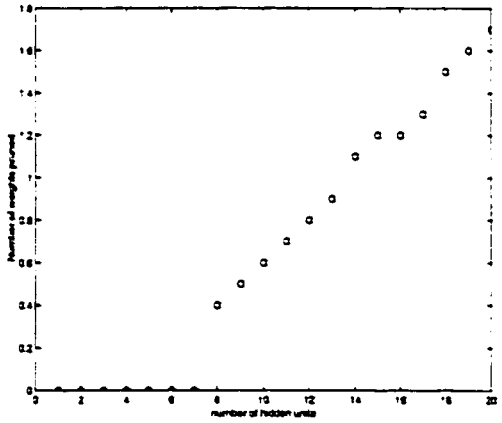
Figure 2.12: (a) Training and (b) generalization errors for CIF with and without pruning method A, and (c) the cumulative number of weights pruned, versus the number of hidden units; solid line: with and dashed line: without pruning method A.



(a) FVU for the training

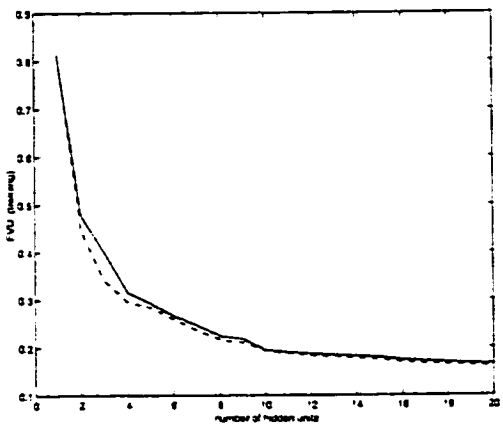


(b) FVUs for the generalization

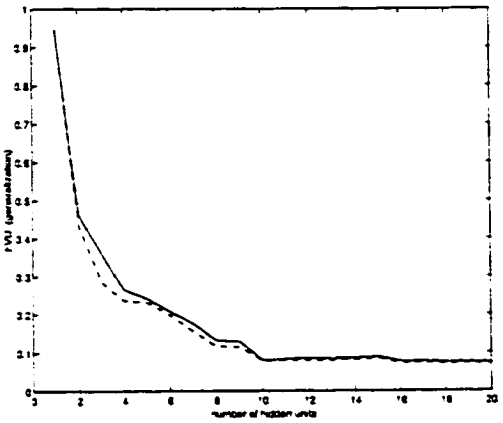


(c) Cumulative number of weights pruned

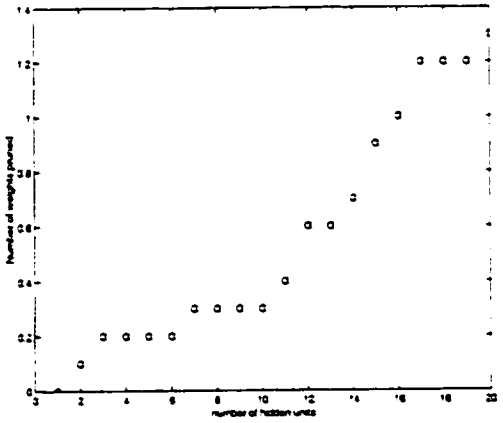
Figure 2.13: (a) Training and (b) generalization errors for CIF with and without pruning method B, and (c) the cumulative number of weights pruned, versus the number of hidden units; solid line: with and dashed line: without pruning method B.



(a) FVUs for the training

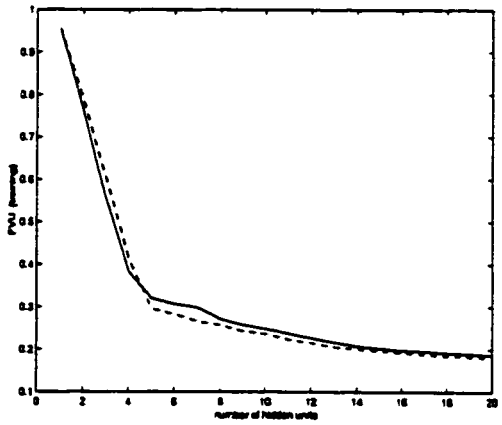


(b) FVUs for the generalization

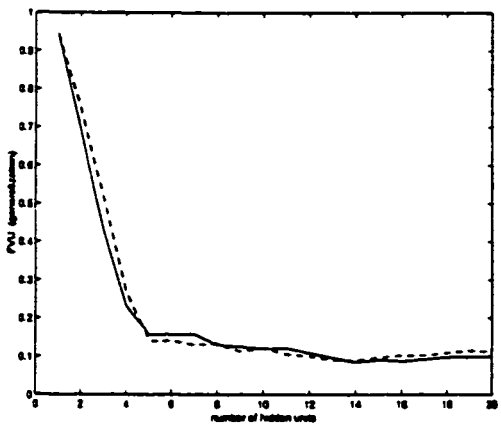


(c) Cumulative number of weights pruned

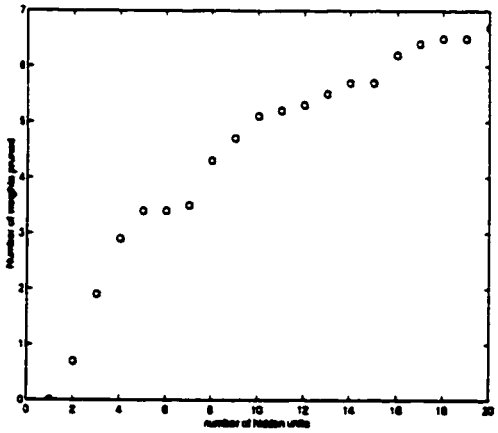
Figure 2.14: (a) Training and (b) generalization errors for SAF with and without pruning method A, and (c) the cumulative number of weights pruned, versus the number of hidden units; solid line denotes with and dashed line denotes without pruning method A.



(a) FVUs for the training

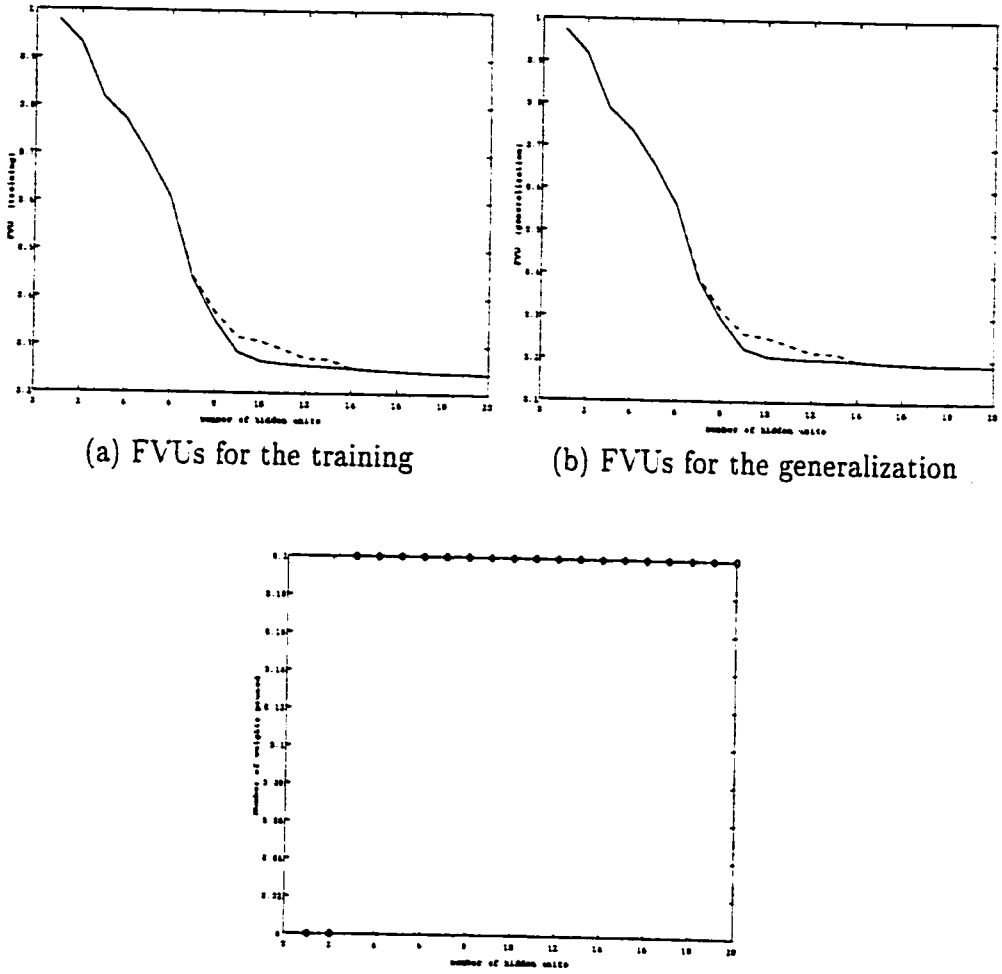


(b) FVUs for the generalization



(c) Cumulative number of weights pruned

Figure 2.15: (a) Training and (b) generalization errors for SAF with and without pruning method B, and (c) the cumulative number of weights pruned, versus the number of hidden units; solid line: with and dashed line: without pruning method B.

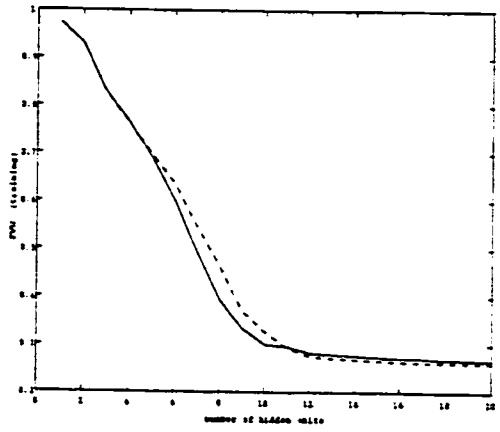


(a) FVUs for the training

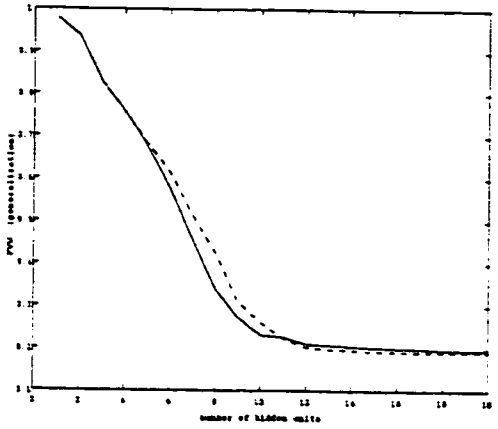
(b) FVUs for the generalization

(c) Cumulative number of weights pruned

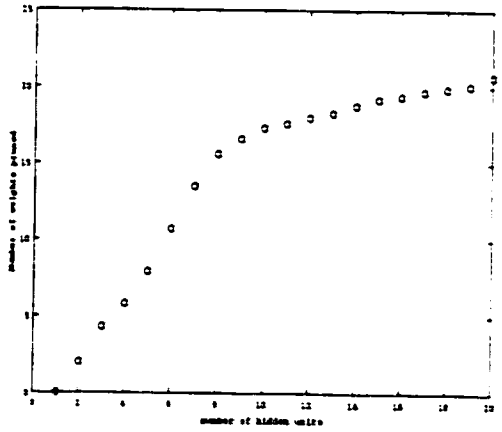
Figure 2.16: (a) Training and (b) generalization errors for HAS with and without pruning method A, and (c) the cumulative number of weights pruned, versus the number of hidden units; solid line denotes with and dashed line denotes without pruning method A.



(a) FVUs for the training



(b) FVUs for the generalization



(c) Cumulative number of weights pruned

Figure 2.17: (a) Training and (b) generalization errors for HAS with and without pruning method B, and (c) the cumulative number of weights pruned, versus the number of hidden units; solid line denotes with and dashed line denotes without pruning method B.

2.4 Convergence properties for special class of constructive algorithms

For the constructive OHL-FNNs, the convergence of the algorithm with respect to the number of hidden units is very important and needs careful investigation.

First, we investigate the ideal case where a "perfect" input-side training is assumed to determine the convergence of the constructive training algorithm with and without error scaling operations. The ideal case can yield the best convergence rate that the constructive training algorithm can theoretically approach. Furthermore, through this analysis one obtains a better understanding as to the implications for a "perfect" or ideal input-side training.

Secondly, the convergence properties of the nominal case for the constructive algorithm where a "perfect" input-side training is not achievable will be considered. Formal analysis of the input-side training is not tractable due to the nonlinearities of the activation functions of the hidden units. Fortunately, due to the linearity of the output node of the network, one may formally analyze the output-side training to determine how the input-side training affects the output-side training, and further to determine how the output error changes with the addition of new hidden units as the constructive learning proceeds. Consequently, a sufficient condition is developed in this section for the input-side training to guarantee the convergence of the constructive algorithm. This condition also provides us with a formal tool to better understand the previously designed objective functions in [45]. Furthermore, guidelines for the design of the objective function to be utilized for the input-side training may also be obtained.

2.4.1 Convergence properties for an ideal case

After the initial network with, say one or more number of hidden units, has been constructed using a BP-based or some second-order training algorithm, the output error e_{n-1}^j for the $n-1$ hidden unit(s) can be calculated by

$$\begin{aligned} e_{n-1}^j &= d^j - y_{n-1}^j \\ &= d^j - \sum_{k=0}^{n-1} v_k f_k(s_k^j), \qquad j = 1, 2, \cdots, P. \end{aligned} \tag{2.37}$$

If one attempts to train a new (namely the n-th) hidden unit added to the network where the input-side training is perfect or ideal, i.e., (2.15) is completely satisfied, then for the output-side training one can set the output-side weight of the new hidden unit v_n as

$$v_n = \frac{1}{\lambda} \tag{2.38}$$

while leaving other existing output-side weights unchanged. Therefore, the output error e_n^j will now become

$$\begin{aligned} e_n^j &= d^j - y_n^j \\ &= d^j - y_{n-1}^j - v_n f_n(s_n^j) \\ &= e_{n-1}^j - \frac{1}{\lambda}\left\{\lambda e_{n-1}^j\right\} \\ &= 0 \end{aligned} \tag{2.39}$$

for all j, implying that the constructive OHL network requires to add only one hidden unit to the initial network in order to achieve zero output error. However since (2.15) can not be satisfied in general, therefore the above ideal situation is highly unlikely to occur. Below we provide some further explanations for this observation.

If (2.15) holds, then we would have

$$\begin{aligned} w_{n,1}x_1^1 + w_{n,2}x_2^1 + \cdots + w_{n,M}x_M^1 + w_{n,0} &= f^{-1}(\lambda e_{n-1}^1) \\ &\vdots \\ w_{n,1}x_1^j + w_{n,2}x_2^j + \cdots + w_{n,M}x_M^j + w_{n,0} &= f^{-1}(\lambda e_{n-1}^j) \\ &\vdots \\ w_{n,1}x_1^P + w_{n,2}x_2^P + \cdots + w_{n,M}x_M^P + w_{n,0} &= f^{-1}(\lambda e_{n-1}^P) \end{aligned} \tag{2.40}$$

where $f^{-1}(\cdot)$ is the inverse of $f(\cdot)$ which is generally a monotonic function. The above set of P simultaneous equations have $M+1$ unknown variables $w_{n,0}, \ldots, w_{n,M}$. If $P = M+1$ and the related matrix rank conditions hold, then (2.15) will be satisfied, implying that the ideal case is feasible. However, in most regression problems P is much larger than $M+1$, and hence only an LS solution may be computed using the above set of simultaneous equations.

When the output error is scaled according to (2.20), for $C_1 < 1$ we may write

$$\begin{aligned} e_n^j &= d^j - y_{n-1}^j - \Delta v_0 - v_n f_n(s_n^j) \\ &= e_{n-1}^j - \Delta v_0 - v_n \lambda (C_1 e_{n-1}^j + C_2) \\ &= (1 - v_n C_1 \lambda) e_{n-1}^j - (\Delta v_0 + v_n C_2 \lambda). \end{aligned} \tag{2.41}$$

where Δv_0 denotes the change in the bias of the output node. The other set of output-side weights $v_1, v_2, \cdots, v_{n-1}$ are left unchanged. If one sets

$$v_n = \frac{1}{\lambda C_1}, \quad \textit{and} \tag{2.42}$$

$$\Delta v_0 = -\frac{C_2}{C_1} \tag{2.43}$$

the error signal e_n^j will be zero for all j. For $C_1 \geq 1$, we have

$$\begin{aligned} e_n^j &= d^j - y_{n-1}^j - \Delta v_0 - v_n f_n(s_n^j) \\ &= e_{n-1}^j - \Delta v_0 - v_n \lambda \left\{ e_{n-1}^j - \bar{e}_{n-1} + \frac{f_{max} + f_{min}}{2} \right\} \\ &= (1 - v_n \lambda) e_{n-1}^j - \left\{ \Delta v_0 - v_n \lambda \bar{e}_{n-1} + v_n \lambda \frac{f_{max} + f_{min}}{2} \right\}. \end{aligned} \tag{2.44}$$

Therefore, by setting

$$\begin{aligned} v_n &= \frac{1}{\lambda}, \ and \\ \Delta v_0 &= \bar{e}_{n-1} - \frac{f_{max} + f_{min}}{2} \end{aligned} \tag{2.45}$$

the output error is forced to zero for all j. Consequently, using the error scaling technique, if (2.15) holds, perfect training (i.e., zero output error) is also possible by properly selecting the bias of the output node and the output-side weights of only the new hidden unit that is added to the initial network.

In practice, as mentioned above, (2.15) is only an ideal case. One is then required to search and obtain an approach by which the output of the new hidden unit approaches as closely as possible to (2.15). This goal is basically equivalent to designing a criterion for the development of the input-side training. The correlation-based criterion (2.10) introduced by Fahlman and Lebiere [42] may be viewed and considered as one such design.

2.4.2 Convergence properties of an improved constructive algorithm

In [45], a new objective function or criterion for input-side training was derived as:

$$J = \frac{\left(\sum_{j=1}^{P} e_{n-1}^{j} f_n(s_n^j) \right)^2}{\sum_{j=1}^{P} \left(f_n(s_n^j) \right)^2} \tag{2.46}$$

It was concluded that this function compared to the other objective functions used for input-side training of a OHL-FNN results in somewhat better performance in solving some regression problems [45]. 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 not 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 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.

Suppose the output weights of the hidden units are denoted by $\mathbf{V}_{n-1} = (v_0, v_1, v_2, \cdots, v_{n-1})^T$ before the n-th 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 n-th 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 n-th hidden unit is now given by

$$\begin{aligned} SSE_n &= \sum_{j=1}^{P}(d^j - y_n^j)^2 \\ &= \sum_{j=1}^{P}\left\{d^j - (\mathbf{V}_{n-1}^T + \Delta\mathbf{V}_{n-1}^T)\mathbf{f}_{n-1}^j - v_n f_n(s_n^j)\right\}^2 \\ &= \sum_{j=1}^{P}\left(e_{n-1}^j - \mathbf{U}_n^T\mathbf{f}_n^j\right)^2 \end{aligned} \tag{2.47}$$

where

$$\mathbf{U}_n = (\Delta\mathbf{V}_{n-1}^T, v_n)^T, \tag{2.48}$$

$$\mathbf{f}_n^j = \left(f_0, f_1(s_1^j), \cdots, f_{n-1}(s_{n-1}^j), f_n(s_n^j)\right)^T, \tag{2.49}$$

and note that $f_0 = 1$. After some algebraic manipulations, equation (2.47) may be written as

$$SSE_n = SSE_{n-1} - 2\mathbf{U}_n^T\mathbf{Q}_n + \mathbf{U}_n^T\mathbf{R}_n\mathbf{U}_n \tag{2.50}$$

where

$$\mathbf{Q}_n = \sum_{j=1}^{P} e_{n-1}^j\mathbf{f}_n^j, \tag{2.51}$$

$$\mathbf{R}_n = \sum_{j=1}^{P}\mathbf{f}_n^j(\mathbf{f}_n^j)^T. \tag{2.52}$$

Minimization of the above SSE_n leads to the following LS solution

$$\mathbf{U}_n = \mathbf{R}_n^{-1}\mathbf{Q}_n \tag{2.53}$$

and the minimum value is given by

$$SSE_n^{opt} = SSE_{n-1} - \mathbf{Q}_n^T\mathbf{R}_n^{-1}\mathbf{Q}_n. \tag{2.54}$$

Obviously, maximizing the 2nd term in the RHS of the above equation will maximize the convergence in minimizing the SSE (or MSE). Clearly, this gives us a very 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 (2.54) positive, then 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 above term is positive and possibly maximized. Since the 2nd term is highly nonlinear, and the many previously proposed objective functions in [45] 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, the optimum objective function may be given by

$$J_{input,opt} = \mathbf{Q}_n^T\mathbf{R}_n^{-1}\mathbf{Q}_n \tag{2.55}$$

which is different from the objective function J derived in [45]. 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 required 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 (2.10) is included in $J_{input,opt}$, and maximization of this correlation will

contribute to making $J_{input,opt}$ 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.

By letting

$$\mathbf{Q}_n^T = [\mathbf{Q}_{n-1}^T, Q_n], \tag{2.56}$$

$$Q_n = \sum_{j=1}^{P} e_{n-1}^j f_n(s_n^j), \tag{2.57}$$

$$\mathbf{R}_n = \begin{bmatrix} \mathbf{R}_{n-1} & \mathbf{r}_{n-1} \\ \mathbf{r}_{n-1}^T & r_n \end{bmatrix}, \tag{2.58}$$

$$\mathbf{r}_{n-1} = \sum_{j=1}^{P} f_n(s_n^j) \mathbf{f}_{n-1}^j, \tag{2.59}$$

$$r_n = \sum_{j=1}^{P} \left(f_n(s_n^j)\right)^2 \tag{2.60}$$

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

$$SSE_n^{opt} = SSE_{n-1} - \mathbf{Q}_{n-1}^T \mathbf{R}_{n-1}^{-1} \mathbf{Q}_{n-1} - Q_n^2 / r_n. \tag{2.61}$$

It now follows that minimization of SSE_n yields the objective function J derived in [45] (the 3rd term in the RHS of (2.61)). 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 is derived is an optimal objective function that does include the objective function derived in [45] as its special case. Implementing or developing a network structure that would realize our derived objective function requires further research.

2.5 Conclusions

In this chapter, a new technique is proposed to scale the error signal during the constructive learning process to improve the input-side training efficiency and to obtain better generalization performance. Two pruning methods for removing the input-side redundant connections have also been proposed. The result would be a smaller network without degrading or compromising its performance. Finally, convergence of the proposed constructive algorithm has also been discussed and an optimum objective function for input-side training was subsequently derived.

Chapter 3

A new strategy for constructing multi-hidden-layer FNNs

3.1 Introduction

It was stated in previous chapters that one-hidden-layer (OHL) network can approximate any continuous function to any desired accuracy as long as enough hidden units are included in the hidden layer [34]. Motivated by this result the developments in the literature have been focused on constructive OHL-FNNs. However, the result in [34] does not necessarily imply that a OHL network architecture is the most efficient structure for representing a given input-output map. Networks with more than one hidden layer usually perform better than the OHL networks in many applications.

Constructing and training multi-hidden-layer networks is generally more difficult than those of OHL networks since one has to develop a strategy by which the depth of the network in terms of the number of layers is determined. Cascade-Correlation (CC) constructs a multi-hidden-layer FNN, but the connections in the resulting network are not regular. As described in Chapter 1 there are many limitations with CC. Other algorithms, such as the stack learning algorithm by Fang and Lacher [47] constructs similar connections as in CC, and the adding-and-deleting

algorithm by Nabhan and Zomaya [48] which is based on an exhaustive search technique also produce multi-hidden-layer FNNs that are expected to work better than the OHL-FNNs. However, the network connections may be irregular and the training process may also require a long time to converge.

In this chapter, a new strategy for constructing a multi-hidden-layer FNN with regular connections is proposed by extending the strategy that was developed in constructive OHL-FNNs [45]. 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 CC and the stack learning algorithm has been eliminated. In other words, the new constructive network scales up appropriately.

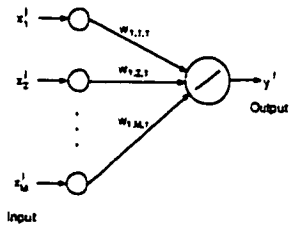
(3) The algorithm adds hidden units and hidden layers one at a time, when and if they are needed. The input-side weights for all the hidden units in the installed hidden layer are fixed (input weights are frozen), however, the input-side weights for a new hidden unit that is being added to the network are updated during the input-side training. Generally, a constructive OHL may need to grow a large number of hidden units or even fail to represent a map that actually requires more than a single hidden layer to handle the corresponding complexity. Therefore, the proposed algorithm that grows its layers and hidden units automatically depending on the complexity of the problem being considered will converge very fast and is computationally more efficient as compared to the constructive OHL networks.

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

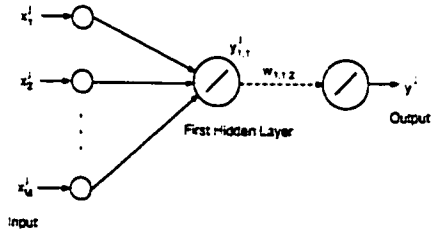
The outline of this chapter is as follows. Section 3.2 presents the constructive multi-hidden-layer strategy. Simulation results are given in Section 3.3. Section 3.4 presents our conclusions.

3.2 Strategy for constructing multi-hidden-layer FNNs

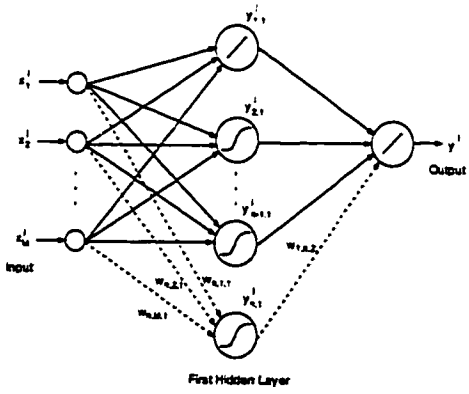
In this section, the new strategy for constructing a multi-hidden-layer FNN is presented. Suppose one desires to construct a FNN to solve a mapping or a regression problem having multi-dimensional input space and a single scalar output. Extension to the case of multi-dimensional output space would be straightforward. It is also assumed that the output unit has a linear activation function for the regression problems. The new constructive algorithm develops the FNN (see Figure 3.1) according to the following steps:



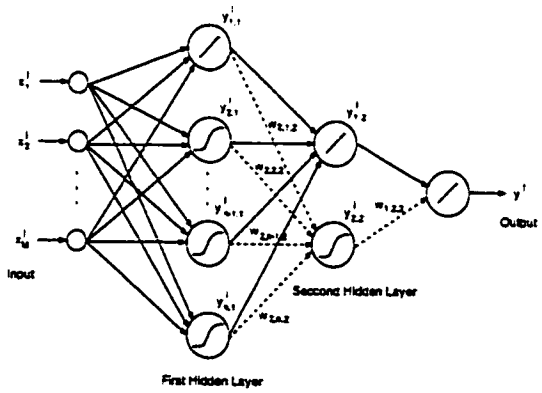
(a) Initial linear network



(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

Figure 3.1: Multi-hidden-layer network construction process for the proposed strategy.

Step 1: Initially train the network with a single output node and no hidden layers (refer to Figure 3.1(a)), with the input belonging to a multi-dimensional space depending on the problem that is being considered. The training of the weights may be achieved according to an LS solution or some gradient-based algorithm, including the bias of the output node. The training error FVU (as described in Section 2.2.2) is monitored each time a new hidden unit is added to the network, and if it is not smaller than some prespecified required threshold, one would then proceed to Step 2a, otherwise one would go to Step 2c.

Step 2a: Treat the output node as a hidden unit with unit output-side weight and zero bias (refer to Figure 3.1(b)). The hidden unit input-side weights are fixed and are as determined from the previous training step. The output error of the new linear output node will be identical to its hidden unit.

Step 2b: A new hidden unit is added one at a time just as in a constructive OHL-FNN (refer to Figure 3.1(c)). Each time a new hidden unit is added, the output error (FVU) is monitored to see if it is smaller than the 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. Under this condition (when error is stabilized) one then proceeds to Step 2a; a new hidden layer is introduced to possibly further reduce the output error (refer to Figure 3.1(d)). Steps 2a and 2b are repeated as many times as necessary until the output error FVU is below the prescribed desired threshold. Input training of a new hidden unit is performed by maximizing a correlation-based objective function, whereas the output training is performed through an LS type algorithm. Subsequently, one then proceeds to Step 2c.

Step 2c: The constructive training is terminated.

For implementing the above algorithm some further details and definitions regarding the above procedure are in order. First, the quantity "stop-ratio" is introduced to determine if the training error FVU is stabilized as a function of the added hidden unit, as follows

$$stop-ratio = \frac{FVU(n-1) - FVU(n)}{FVU(n-1)} \times 100\% \tag{3.1}$$

where $FVU(n)$ indicates the training error FVU when the n-th hidden unit is added to a growing hidden layer of the network. Note that according to [34] addition of a proper hidden unit to an existing network will reduce the training error. The issue that needs addressing here is the magnitude 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 becomes "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, since the unnecessary hidden units may force the network to begin to memorize the data. 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 "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 here that the stop-ratio and the stop-num are 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 determining these parameters.

C1: A rule of thumb, say 3 to 10 percent, should be a reasonable value 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 network with fewer hidden layers is clearly more advantageous over networks

with many hidden layers. Therefore, this ratio may initially be set to relatively small value 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 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 ratio in BP networks).

C2: The stop-num values larger 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 the 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 other subsequent layers, say 3 or even 2.

Note here that in the proposed algorithm the policy for hidden unit addition is the same as that adopted in Section 2.2. Therefore, the convergence analysis presented previously is also applicable and valid here.

Before the experimental results are introduced, let us explain in more details why the proposed constructive algorithm can not automatically be extended to classification problems where the output node has nonlinear activation function. As shown in Figure 3.2, 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, we have

$$y^j_{1,L+1} = g(\beta y^j_{1,L} + \alpha), \quad j = 1, 2, \cdots, P \tag{3.2}$$

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 obtain

$$y^j_{l,L+1} = y^j_{l,L}, \quad j = 1, 2, \cdots, P \tag{3.3}$$

This implies that the training error from the previous output node will be passed

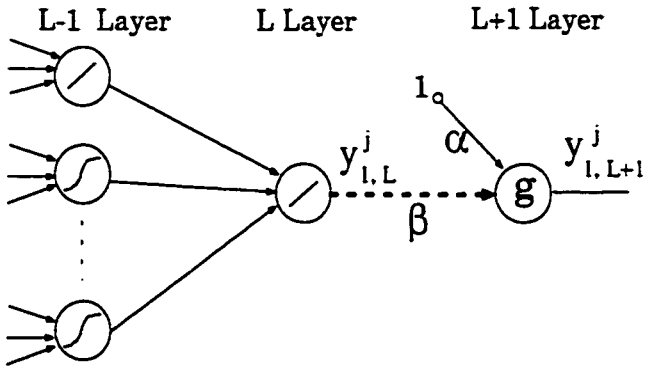


Figure 3.2: Multi-hidden-layer network for the proposed strategy.

on to the newly generated output node unchanged as is. When a new hidden unit is added to the L-th 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 satisfy the following:

$$\beta y^j_{l,L} + \alpha = g^{-1}(y^j_{l,L+1}) = g^{-1}(y^j_{l,L}) \tag{3.4}$$

for some j. For serial learning or a single training sample, (3.4) 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 thesis, we need to ensure that the following set of equations hold simultaneously:

$$\begin{aligned} \beta y_{1,L}^1 + \alpha &= g^{-1}(y_{1,L}^1) \\ \beta y_{1,L}^2 + \alpha &= g^{-1}(y_{1,L}^2) \\ &\vdots \\ \beta y_{1,L}^P + \alpha &= g^{-1}(y_{1,L}^P) \end{aligned} \tag{3.5}$$

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

$$\begin{bmatrix} \beta \\ \alpha \end{bmatrix} = \left(A^T A\right)^{-1} A^T B \tag{3.6}$$

where

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

Note that in the above expression it is implicitly assumed that matrix $A^T A$ is full rank. In most problems where a large number of training samples are available, the rank condition is generally satisfied easily. The summed square error of the above simultaneous equations is given by

$$\begin{aligned} J &= \frac{1}{2}\sum_{j=1}^{P}\left\{\beta y_{1,L}^j + \alpha - g^{-1}(y_{1,L}^j)\right\}^2 \\ &= \frac{1}{2}\left(A\begin{bmatrix} \beta \\ \alpha \end{bmatrix} - B\right)^T \left(A\begin{bmatrix} \beta \\ \alpha \end{bmatrix} - B\right) \\ &= \frac{1}{2}B^T\left[I - A\left(A^T A\right)^{-1} A^T\right] B \;>\; 0 \end{aligned} \tag{3.7}$$

which is always positive. 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

$$SSE(L+1) = \sum_{j=1}^{P} \left\{ d^j - g(\beta y_{1,L}^j + \alpha) \right\}^2 \tag{3.8}$$

where d^j is the desired target.

From the above analysis, it now follows clearly 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 result. On the other hand, if the addition of hidden units to the previous layer can not 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 for some applications. This issue will not be pursued further in this thesis.

3.3 Simulation results for regression problems

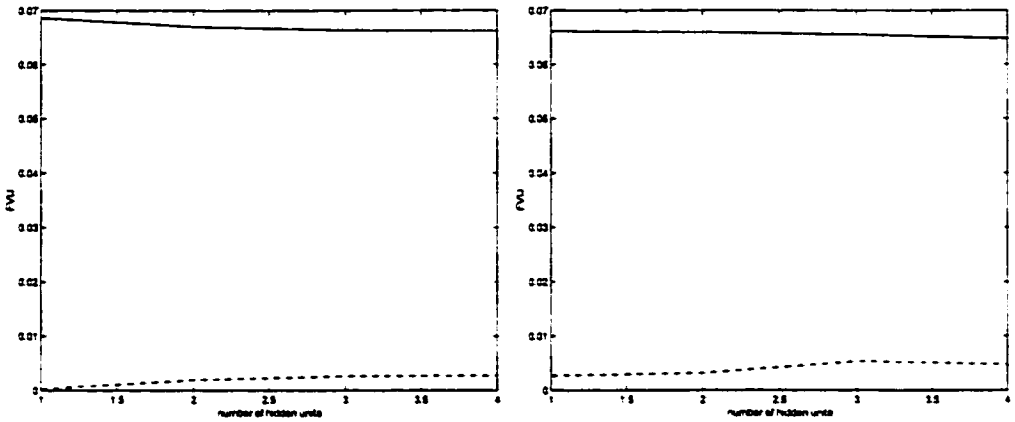
To confirm the effectiveness and potential of the proposed strategy, we have conducted several simulations for the regression problems including those considered previously in Chapter 2. Unless explicitly specified, for all the simulation results shown below $stop - ratio = 5\%$ and $stop - num = 3$ during the entire construction process.

The first problem considered is a linear regression given by

$$g(x) = 0.75x + 0.25 + v, \quad 0 < x < 1. \tag{3.9}$$

One hundred (100) samples are used for training. The SNR is 10 [dB]. Figure 3.3 shows the training and the generalization error FVUs associated with the first and the second hidden layers. Figure 3.3(a) indicates that the addition of a 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

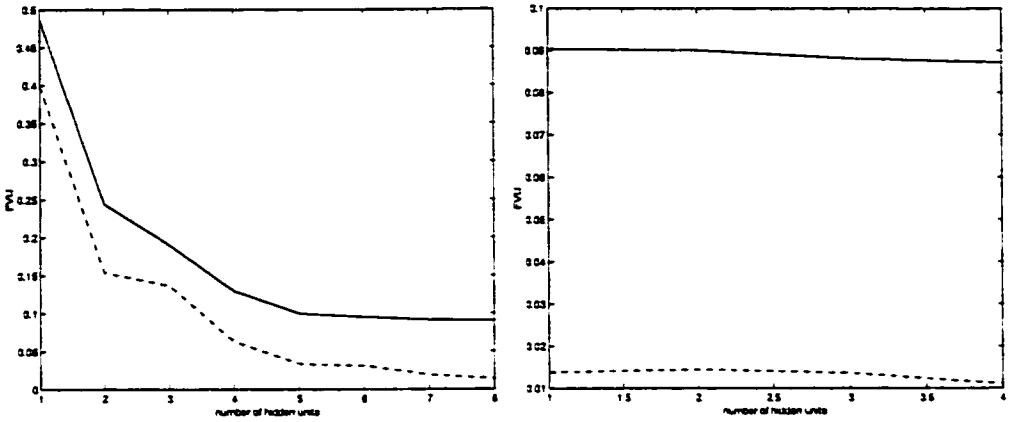
added. This implies that all the three nonlinear hidden units added to the network are redundant and only the linear unit is needed, resulting in a linear network for representing a linear regression problem. To observe the possibility of reducing the FVU by using a deeper network, the second hidden layer was also added. Figure 3.3(b) depicts the FVUs. Obviously, the saturation of the training FVU is more pronounced in this case.



(a) First hidden layer (b) Second hidden layer

Figure 3.3: Training (solid line) and generalization (dashed line) FVUs of a multi-hidden-layer network constructed for a simple linear regression problem.

The second problem 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 each constructed hidden layers are depicted in Figure 3.4. It follows from Figure 3.4(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 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.

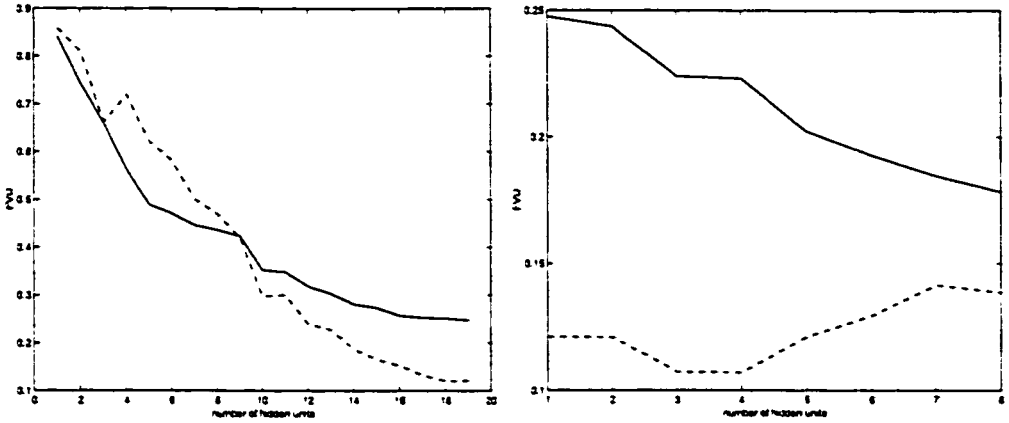


(a) First hidden layer (b) Second hidden layer

Figure 3.4: Training (solid line) and generalization (dashed line) FVUs of a multi-hidden-layer network constructed for a sinusoid with noise problem.

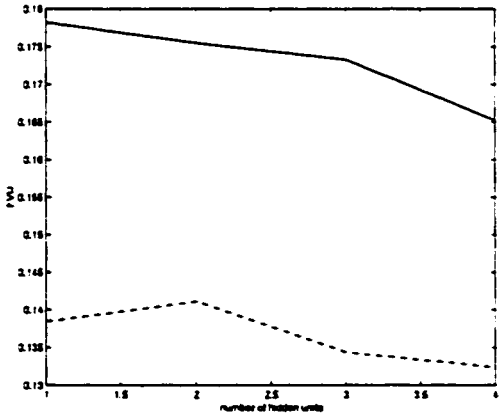
The third problem considered here is the CIF that was simulated in Chapter 2. The SNR is selected about 6 [dB]. Two hundred and twenty five (225) data points are used to train a network. The results are given in Figure 3.5. Clearly, it can be observed that the second hidden layer is needed in this case, however the third hidden layer does not contribute much to solving this problem (in terms of 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 Figure 3.5(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 are shown along with its original surface in Figure 3.6.

The last problem we consider here is the SAF problem with noise. Two hundred (200) samples are used to train a network. The SNR is selected to be 10 [dB].



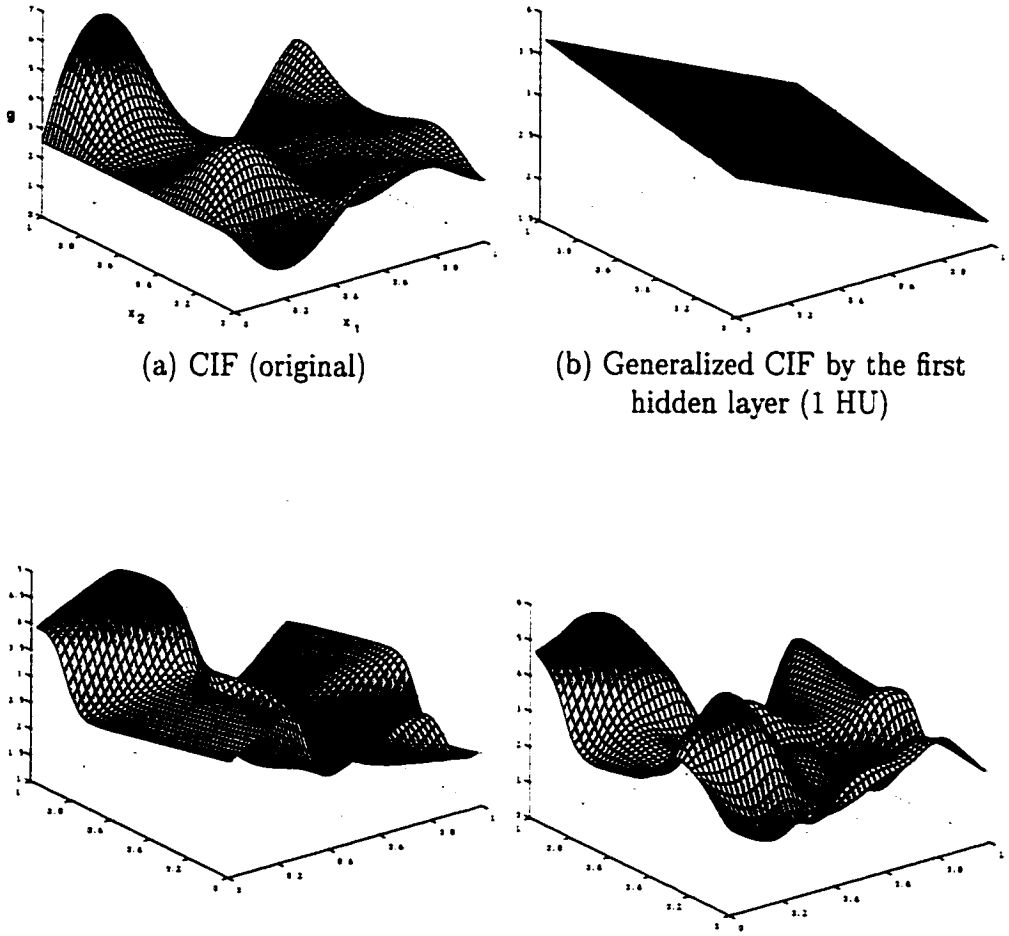
(a) First hidden layer

(b) Second hidden layer



(c) Third hidden layer

Figure 3.5: Training (solid line) and generalization (dashed line) FVUs of a multi-hidden-layer network constructed for CIF.



(a) CIF (original)

(b) Generalized CIF by the first hidden layer (1 HU)

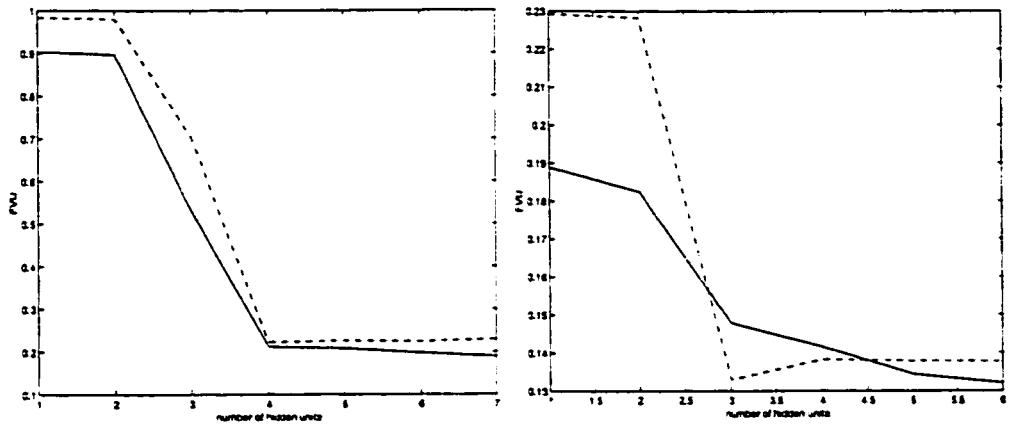
(c) Generalized CIF by the first hidden layer (5 HUs)

(d) Generalized CIF by the first hidden layer (15 HUs)

Figure 3.6: Original and generalized CIF by the first hidden layer.

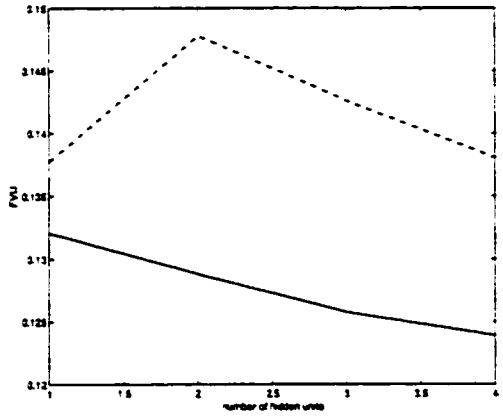
The incentive given to the first hidden layer is by setting its stop-ratio to 5%, and for the other hidden layers this parameter is set to 10%. Clearly, from Figure 3.7 it follows that a two-hidden-layer network is all that is required for solving the SAF problem, as the 3rd layer causes the stop-ratio 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 the new algorithm has constructed a network having simultaneously a sufficient number of hidden layers and a sufficient number of hidden units in each hidden layer depending on the complexity of the problem that is being considered.



(a) First hidden layer

(b) Second hidden layer



(c) Third hidden layer

Figure 3.7: Training (solid line) and generalization (dashed line) FVUs of a multi-hidden-layer network constructed for SAF.

3.4 Conclusions

In this chapter, a new strategy for constructing a multi-hidden-layer FNN was presented. The new algorithm builds upon the constructive algorithm developed for adding hidden units to a OHL network by generating additional hidden layers. The network obtained by using this algorithm has regular connections to facilitate hardware implementation. The constructed network is suboptimal in the sense that the new hidden units and layers are added incrementally only when they are needed based on monitoring the residual error that can not be reduced any further by the already established network. It is not feasible to compare the new algorithm with the other existing multi-hidden-layer constructive algorithms reported in the literature, due to the fundamental differences in the network architecture and construction strategies. However, it is seen from the presented experimental results that the proposed algorithm is effective and efficient in representing certain nonlinear problems.

Chapter 4

Constructive one-hidden-layer FNNs using polynomial activation functions

4.1 Introduction

Since the early 1980s, a large number of neural network (NN) structures have been proposed and applied to various real world problems. The feedforward NNs (FNNs) are by far the most popular architectures due to their structural flexibility, good representational capabilities, and availability of a large number of training algorithms. The hidden units in a FNN usually have the same activation functions typically selected as sigmoidal functions.

Among the constructive FNNs, the one-hidden-layer FNN (OHL-FNN) is by far the simplest in terms of both its structure and training efficiency, yet having a wide applicability due to its "universal approximation property". That is, a OHL-FNN can approximate a continuous function to any desired accuracy as long as enough hidden units with sigmoidal activation functions are included. However, it has never been shown that the use of the same activation functions for all the

hidden units is the best or the optimal choice for performance and generalization considerations. In other words, there are still a lot of opportunities for attempting to possibly improve the performance of the OHL-FNN by using more appropriate activation functions for the hidden units instead of simply using identical sigmoidal functions.

In a constructive OHL-FNN, one or a pool of candidates with different initial weights and possibly different activation functions are tested and the one resulting in maximizing the performance index will be incorporated into the active network. However, the freedom in actually determining and selecting the activation functions will significantly increase the search space due to the difficulty in specifying the pool of activation functions that may be used. Although, the idea of using different activation functions for different units was mentioned by Fahlman [42] and several other researchers [45, 34], a systematic and a rigorous algorithm and a methodology for accomplishing this has not yet been developed.

In this part of the thesis, a new strategy is developed and presented that is applicable to both fixed structure as well as the constructive network trainings. This is accomplished by using different activation functions with hierarchically varied nonlinearities, as the constructive learning of a OHL-FNN is progressing. This is motivated by the idea that a non-uniform use of activation functions may actually enhance the generalization capability of the resulting network.

Let us consider an input-output map to be realized by a OHL-FNN. This function may generally contain a constant term, a linear term, second order nonlinear terms and higher-order nonlinear terms. Any analytic function can be represented in this way at least in the sense of its Taylor series expansion. All these terms may in the end be viewed as being combined in "parallel", in the sense that their weighted sum would amount to the whole function. It is quite well-known that any series expansion can be made to approximate the original function to any desired level of accuracy, as long as sufficient number of terms are used. This well-known fact, in

principle, is very much analogous to the above-mentioned approximation ability of a FNN. The idea is to basically let each unit in the FNN represent only one term of the function representation that is being approximated (along the similar lines as in the Fourier or the Taylor series expansions). Towards this end, we propose to use locally bounded orthonormal basis functions as the activation functions of the hidden units of a FNN. Each unit is expected to be responsible for approximating the corresponding nonlinearity contained in the underlying function. Note that simply using the nonlinear functions in a Taylor series expansion can not be used as the activation functions of the hidden units since these functions are not bounded in general.

In this chapter, the incremental constructive structure of FNNs when applied to regression problems is considered. A OHL-FNN with a linear output layer is utilized here. Unlike the constructive learning approach proposed in [45], the initial network has no hidden units. The network is built up from a so-called "null net". In [45], the number of hidden units for the initial network was left unsolved and to be resolved by trial and error and heuristics. During the construction process, the hidden units are added to the active network one at a time, and the activation functions of the hidden units will be assigned from lower order orthonormal polynomials to higher-order ones. As the network grows, naturally the activation functions of the hidden units become more complicated. This can potentially cause some problems in practice, since more computational resources will be clearly needed. However, if one utilizes orthonormal polynomials that have recursive characteristics, this problem can be alleviated to a large extend. For example, the orthonormal Hermite polynomials are one such polynomial candidates that may be considered, since they have the following desirable properties [49]: (a) any analytic signal/function (or objective signal/function in a regression problem) can be represented to an arbitrarily high degree of accuracy by taking sufficient number of basis functions in a series expansion, implying that the orthonormal Hermite polynomials may be used as universal

approximators; and (b) the recursive expressions for calculating the basis functions and the recursive relationships for their first-order derivatives can be utilized effectively in network training. Furthermore, extensive simulations for many noisy regression problems have revealed that the proposed scheme may produce FNNs that generalize much better than constructive OHL-FNNs with identical sigmoidal activation functions [45].

The outline of this chapter is as follows. Section 4.2 presents a brief introduction to the Hermite polynomials. An incremental training algorithm is developed in Section 4.3. Simulation results are provided in Section 4.4 to demonstrate the effectiveness and potentials of the new constructive network. The conclusions are briefly drawn in Section 4.5.

4.2 Hermite polynomials

In this section, the Hermite polynomials having hierarchical nonlinearities are first introduced to be used subsequently as the activation functions of the hidden units of the FNN. The orthogonal Hermite polynomials defined over $(-\infty, \infty)$ are given as follows [49] (see also the references therein):

$$H_0(x) = 1, \tag{4.1}$$

$$H_1(x) = 2x, \tag{4.2}$$

$$\vdots$$

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x), \ n \geq 2. \tag{4.3}$$

The definition of $H_n(x)$ may be given alternatively by

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n}\left(e^{-x^2}\right), \quad n > 0, \quad H_0(x) = 1. \tag{4.4}$$

The polynomials given in (4.1)–(4.3) are orthogonal to each other, but not orthonormal. The orthonormal Hermite polynomials may then be defined according to

$$h_n(x) = \alpha_n H_n(x)\phi(x), \tag{4.5}$$

where

$$\alpha_n = (n!)^{-1/2}\pi^{1/4}2^{-(n-1)/2}, \tag{4.6}$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}. \tag{4.7}$$

In other words, for a given $h_n(x)$, the following orthonormal relationship holds

$$\int_{-\infty}^{\infty} h_i(x)h_j(x)dx = \begin{cases} 1 & i = j, \\ 0 & i \neq j. \end{cases} \tag{4.8}$$

The first-order derivative of $h_n(x)$ can be easily obtained by virtue of the recursive nature of the polynomials defined in (4.3), that is:

$$\frac{dh_n(x)}{dx} = (2n)^{1/2}h_{n-1}(x) - xh_n(x), \quad n \geq 1, \tag{4.9}$$

$$\begin{aligned} \frac{dh_0(x)}{dx} &= \alpha_0\frac{d\phi(x)}{dx} \\ &= -xh_0(x), \quad n = 0. \end{aligned} \tag{4.10}$$

A typical set of the orthonormal Hermite polynomials are depicted in Figure 4.1. In reference [49], the orthonormal Hermite polynomials are used as basis functions to model 1-D signals in the biomedical field for the purposes of signal analysis and detection. In reference [51], a selected group of the orthonormal Hermite polynomials in a weighted-sum form is used as the activation functions of all the hidden units of the OHL-FNN.

It should be noted here that using the relationships (4.3), (4.9) and (4.10), the computational burden for evaluating the polynomials and their derivatives may be reduced considerably.

In our proposed FNN, the first hidden unit will take $h_0(x)$ as its activation function, the 2nd hidden unit will take $h_1(x)$ as its transfer function, and so on. The resulting FNN with the Hermite polynomials as its activation functions has the same structure as the standard FNN. However, the use of the hierarchical polynomials is expected to increase the performance of the resulting network as shown subsequently in this chapter.

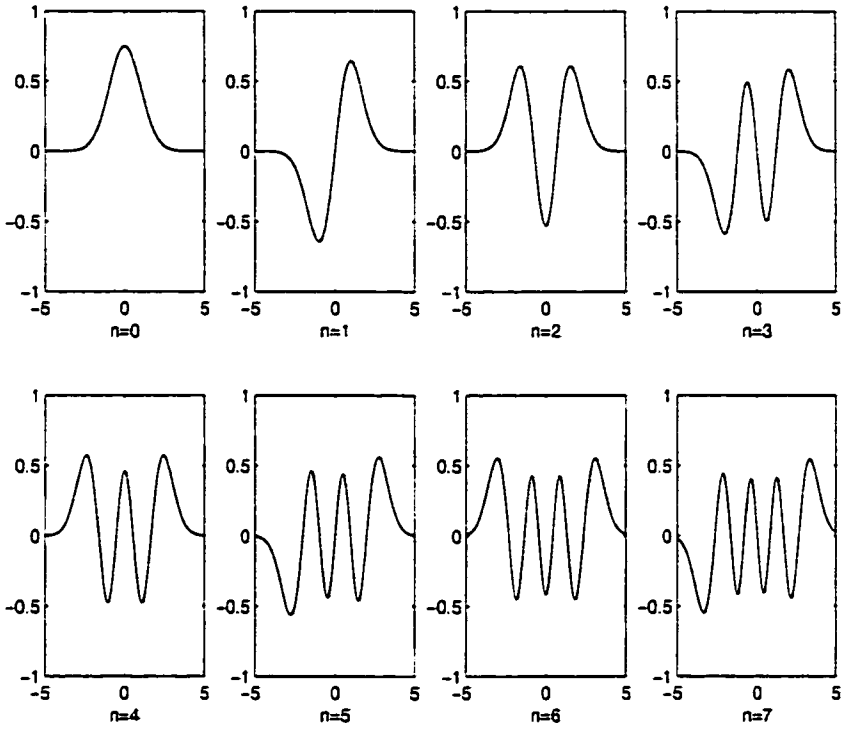


Figure 4.1: Samples of the orthonormal Hermite polynomials, $h_n(\cdot)$ $(n = 0, 1, \cdots, 7)$.

It was indicated earlier that the Hermite polynomials are chosen due to their suitable properties. However, there are many other polynomials that have similar properties. For example, considering the following:

Legendre polynomials

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n, \quad -1 \le x \le 1 \tag{4.11}$$

with recursive relationships

$$nP_n(x) - (2n-1)xP_{n-1}(x) + (n-1)P_{n-2}(x) = 0, \tag{4.12}$$

$$\begin{aligned}(x^2-1)\frac{dP_n(x)}{dx} &= n[xP_n(x) - P_{n-1}(x)] \\ &= (n+1)[P_{n+1}(x) - xP_n(x)].\end{aligned} \tag{4.13}$$

Gegenbauer polynomials

$$\begin{aligned}P_n^\nu(x) &= \frac{(-1)^n}{2^n}\frac{\Gamma(\nu+\frac{1}{2})\Gamma(n+2\nu)}{\Gamma(2\nu)\Gamma(n+\nu+\frac{1}{2})}\frac{(1-x^2)^{(1/2)-\nu}}{n!} \\ &\quad \times \frac{d^n}{dx^n}[(1-x^2)^{n+\nu-(1/2)}], \quad -1 \le x \le 1\end{aligned} \tag{4.14}$$

with recursive relationships

$$nP_n^\nu(x) = 2\nu[xP_{n-1}^{\nu+1}(x) - P_{n-2}^{\nu+1}(x)], \tag{4.15}$$

$$\frac{dP_n^\nu(x)}{dx} = 2\nu P_{n-1}^{\nu+1}(x) \tag{4.16}$$

and $\Gamma(\cdot)$ is a Gamma function.

Tchebycheff polynomials

$$P_n(x) = \frac{(-1)^n}{(2n-1)!}\sqrt{1-x^2}\frac{d^n}{dx^n}(1-x^2)^{n-(1/2)}, \quad -1 \le x \le 1 \tag{4.17}$$

with recursive relationships

$$P_{n+1}(x) - 2xP_n(x) + P_{n-1}(x) = 0, \tag{4.18}$$

$$(1-x^2)\frac{dP_n(x)}{dx} = nP_{n-1}(x) - nxP_n(x). \tag{4.19}$$

Laguerre polynomials

$$P_n^{\alpha}(x) = \frac{e^x x^{-\alpha}}{n!}\frac{d^n}{dx^n}(e^{-x}x^{n+\alpha}), \quad 0 \leq x < \infty \tag{4.20}$$

with recursive relationships

$$nP_n^{\alpha}(x) + (x - 2n - \alpha + 1)P_{n-1}^{\alpha}(x) + (n + \alpha - 1)P_{n-2}^{\alpha}(x) = 0, \tag{4.21}$$

$$[n \geq 2],$$

$$x\frac{dP_n^{\alpha}}{dx} = nP_n^{\alpha}(x) - (n + \alpha)P_{n-1}^{\alpha}(x). \tag{4.22}$$

Note that the input ranges for the above listed polynomials do not meet the requirement for a hidden activation function which takes inputs ranging from $-\infty$ to $+\infty$. If one uses a polynomial with a limited input range as an activation function for a hidden unit, one should then restrict the input-side weight space in which an optimal input-side weight vector has to be searched under a given training criterion. The restriction of the input-side weight space is not desirable as it would limit the representational capability of the network. The rationale for using the Hermite polynomials is therefore motivated by their restriction-free input range properties. Of course, other orthonormal polynomials with restriction-free input range may also be used as activation functions for the FNNs. Search for such polynomials is left for further research.

Mathematically speaking, the above listed polynomials may still be used as activation functions of the hidden units provided that one normalizes the network input vector for a hidden unit each time the input-side weight vector of it is updated. Although, it appears that the weight vector is now not constrained, and only the network input vector is normalized by a properly selected large constant, this constant is actually closely related to the weight vector. Therefore, the input to the hidden unit at its input-side training phase will be no longer linear with respect to the weight vector. Consequently, as far as the optimization problem is concerned, the input-side training will now have two types of nonlinearities : one from the input

to the hidden unit, and another due to the activation function of the hidden unit. This optimization is clearly more complicated as compared to the one with only the nonlinearity of the activation function. It is therefore our conclusion that the Hermite polynomials are expected to be the most suitable choice for the activation functions of the hidden units for the problems considered in this thesis.

4.3 The proposed incremental training algorithm

Consider a typical OHL-FNN with a linear output layer and a polynomial-type hidden layer, as shown in Figure 4.2. The hidden unit with input and output connections indicated by dotted lines is the present neuron that is trained before it is allowed to join the other existing hidden units in the network. Suppose, without loss of generality, that a regression problem has an M-dimensional input vector and a scalar one dimensional output. The j-th input and output samples are denoted by $\mathbf{X}^j = (x_0^j, x_1^j, x_2^j, \cdots, x_M^j)$ $(x_0^j = 1)$ and d^j(target), respectively, where $j = 1, 2, \cdots, P$ (P is the number of training data samples). The OHL constructive algorithm starts from a null network without any hidden units. At any given point during the constructive learning process, suppose there are $n-1$ hidden units in the hidden layer, and the n-th hidden unit is being trained before it is added to the existing network (see Figure 4.2).

The output of the n-th hidden unit for the j-th training sample is given by

$$f_n(s_n^j) = h_{n-1}\left(\sum_{m=0}^{M} w_{n,m} x_m^j\right) \tag{4.23}$$

where $h_n(\cdot)$ denotes the n-th order Hermite orthonormal polynomial. Its derivative can be calculated recursively from equation (4.9) and s_n^j is the input to the n-th hidden unit, given by

$$s_n^j = \sum_{m=0}^{M} w_{n,m} x_m^j \tag{4.24}$$

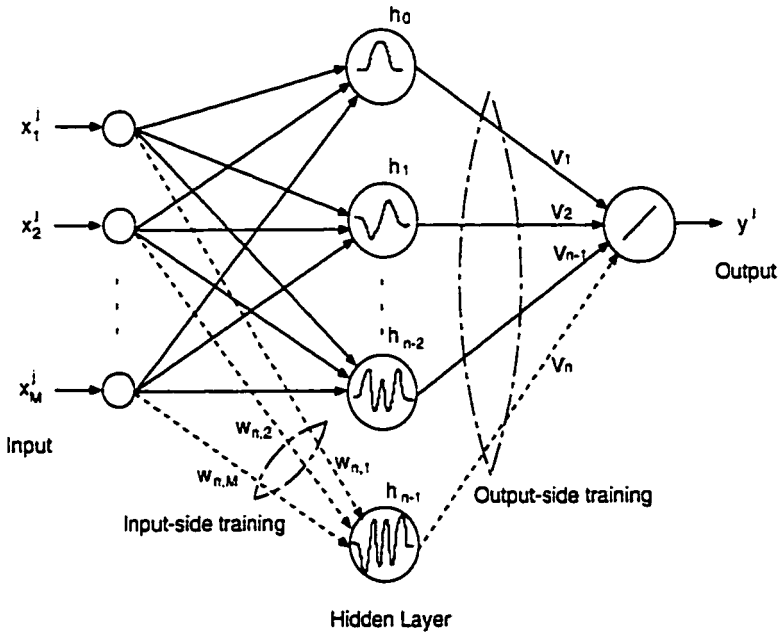


Figure 4.2: Structure of a constructive OHL-FNN that utilizes the orthonormal Hermite polynomials as its activation functions for the hidden units.

where $w_{n,0}$ is the bias of the node and its input is $x_0^j = 1$, and $w_{n,m}$ $(m \neq 0)$ is a weight from the m-th element of the input vector to the n-th hidden node.

The output of the network may be expressed as follows

$$\begin{aligned} y_{n-1}^j &= v_0 + \sum_{i=1}^{n-1} v_i h_{i-1}(s_i^j) \\ &= v_0 + v_1 h_0(s_1^j) + v_2 h_1(s_2^j) + v_3 h_2(s_3^j) + \cdots + v_n h_{n-1}(s_{n-1}^j). \end{aligned} \tag{4.25}$$

where v_0 is the bias of the output node, and $v_1, v_2, \cdots, v_n$ are the weights from the hidden layer to the output node.

Clearly, the above expression has a very close resemblance to a series expansion that utilizes the orthonormal Hermite polynomials as its basis functions, and where each additional term in the expansion contributes to improving the accuracy of the function that is being approximated. Through the series expansion, the approximation error will become smaller as more terms having higher-order nonlinearities are included. This situation is actually quite similar to that of incrementally constructing a OHL-FNN, in the sense that the error is expected to become smaller as new hidden units having hierarchically higher-order nonlinearities are successively added to the network. In fact, it has been shown that under certain circumstances incorporating new hidden units in a OHL-FNN can decrease monotonically the training error [45]. Therefore, in this sense adding a new hidden unit to the network is somewhat equivalent to the addition of a higher-order term in a Hermite polynomial-based series expansion. It is in this sense that the present network is envisaged to be more suitable for constructive learning as compared to a fixed-structure FNNs. Note that, only if $s_1^j = s_2^j = \cdots = s_{n-1}^j$, then y_{n-1}^j will be a truly Hermite polynomial-based series expansion. Otherwise, as proposed in the algorithm above the expansion would be approximate since the weights associated with each added neuron is adjusted separately, resulting in different inputs to the activation functions.

The problem is now to train the n-th hidden unit that is to be added to the active network. There are many objective functions (see for example [42, 34] for

more details) that can be considered for the input-side training of this hidden unit. A simple, but a general cost function that works well, and as discussed in Chapter 2, is as follows [42]:

$$J_{input} = \left| \sum_{j=1}^{P} (e_{n-1}^{j} - \bar{e}_{n-1})(f_n(s_n^j) - \bar{f}_n) \right| \tag{4.26}$$

where

$$f_n(s_n^j) = h_{n-1}(s_n^j) \tag{4.27}$$

$$\bar{e}_{n-1} = \frac{1}{P} \sum_{j=1}^{P} e_{n-1}^{j}, \tag{4.28}$$

$$\bar{f}_n = \frac{1}{P} \sum_{j=1}^{P} h_{n-1}(s_n^j), \tag{4.29}$$

$$e_{n-1}^{j} = d^j - y_{n-1}^{j} \tag{4.30}$$

where $f_n(s_n^j)$ or $h_{n-1}(s_n^j)$ is an orthonormal Hermite polynomial of order $n-1$ used as the activation function of the n-th hidden unit, e_{n-1}^{j} is the network output error when the network has $n-1$ hidden units and d^j is the j-th target output for the network training.

The derivative of J_{input} with respect to a weight $w_{n,i}$ is calculated according to

$$\frac{\partial J_{input}}{\partial w_{n,i}} = sgn\left(\sum_{j=1}^{P} (e_{n-1}^{j} - \bar{e}_{n-1})(f_n(s_n^j) - \bar{f}_n) \right) \times \sum_{j=1}^{P} (e_{n-1}^{j} - \bar{e}_{n-1}) h'_{n-1}(s_n^j) x_i^j \tag{4.31}$$

where $sgn(\cdot)$ is a sign function. The first-order derivative $h'_{n-1}(s_n^j)$ in the above expression can be easily evaluated by using the recursive expression (4.9) for $n \geq 2$ and (4.10) for $n = 1$.

A candidate unit that maximizes the objective function (4.26) will be incorporated into the network as the n-th hidden unit. Subsequently, the output-side training is performed by solving a least squared problem given that the output layer

has a linear activation function (see Figure 4.2). After performing the output-side training, a new error signal e_n^j is calculated for the next cycle of input-side and output-side trainings.

The proposed scheme in this chapter may now be summarized according to the following steps:

Step 1: Initialization of the network

Start the network training process with a OHL-FNN having no hidden units. Set $n = 1$, $e_0^j = d^j$, and the activation function $= h_0(\cdot)$.

Step 2: Input-side training

Train only the input-side weights of the n-th hidden unit $h_n(\cdot)$. The input-side weights for the existing hidden units, if any, are all frozen. This unit will be permanently added to the existing network (active network). Specifically, one candidate for the n-th hidden unit with random initial weights is trained using the objective function defined in (4.26), based on the "quickprop" algorithm [42]. If instead, for the n-th unit, a pool of candidates are trained as above, then the one candidate that yields the maximum objective function will be chosen as the n-th hidden unit to be added to the active network.

Step 3: Output-side training

Train the output-side weights of all the hidden units in the active network. Given that the output layer has a linear activation function the output-side training may be performed by computing the pseudo-inverse operation that results from the least square optimization criterion.

Step 4: Network performance evaluation and training control

Evaluate the network performance in terms of the metric fraction of variance unexplained (FVU) [45] on the training and on the generalization data set

defined as follows

$$FVU = \frac{\sum_{j=1}^{P}(\hat{g}(\mathbf{x}^j) - g(\mathbf{x}^j))^2}{\sum_{j=1}^{P}(g(\mathbf{x}^j) - \bar{g})^2} \tag{4.32}$$

where $g(\cdot)$ is the function being implemented by the FNN, $\hat{g}(\cdot)$ is an estimate of $g(\cdot)$ or the output of the trained network, and $\bar{g}$ is the mean value of $g(\cdot)$. The network training is terminated provided that certain stopping conditions are satisfied. These conditions may be expressed, for example, in terms of a prespecified FVU threshold or a maximum number of permissible hidden units, etc. However, if the stopping conditions are not satisfied, then the network output y_n^j and the network output error $e_n^j = d^j - y_n^j$ are computed, n is increased by 1, i.e., $n = n + 1$, and we then proceed to **Step 2**.

As mentioned previously in Chapter 2, in the constructive OHL-FNNs [45], the determination of a proper initial network size is also a problem that needs careful attention. This initialization may significantly influence the efficiency of the network training that follows. However, in our proposed scheme, the network training starts from the smallest possible architecture. This is an important advantage of our proposed algorithm over the similar methods previously presented in the literature.

Although the proposed algorithm is presented in the context of regression problems, it is straightforward to extend the algorithm to also classification problems by simply changing the activation function of the output node from a linear one to a sigmoidal one. Since the output node is now nonlinear, the pseudo-inverse based LS solution to the output-side training can not be applied any longer. Other algorithms for the nonlinear LS minimization problem has now to be chosen. For instance, second-order algorithms such as the Quasi-Newton algorithm may be utilized.

The proposed algorithm is of incremental nature, and its convergence with respect to the number of hidden units is therefore an important issue that needs to be investigated. Fortunately, this issue can be discussed in quite the same way and context that were taken in Chapter 2, since the only difference between the constructive networks in Chapters 2 and those here is the activation functions utilized in the hidden units. This implies that the same conclusions obtained previously may be extended directly to the constructive networks proposed in this chapter.

4.4 Simulation results

In this section, we present some of our simulated results to demonstrate the effectiveness and the superiority of the new constructive OHL network. It is found through extensive simulations that for "simple" regression problems the performance of the new network is similar to the previous ("standard") constructive OHL networks presented in Chapter 2. The new constructive network, however, provides improved performance when applied to "complicated" regression or classification problems.

In all the following simulations the corresponding learning parameters for both the "standard" OHL constructive algorithm (as in Chapter 2) and the new "proposed" algorithm (as in the present chapter) are set independently so that each network solution would demonstrate its "best" achievable performance on average. Furthermore, it is suggested that this will make the comparisons shown below "fair". Ten (10) independent runs are conducted to obtain an average basis for comparisons.

Example I: Consider the following one-dimensional regression function:

$$g(x) = 0.2\left\{1+\frac{1}{10}(x-7)^2\right\}\cos(2x)+0.5e^{-2x}\sin(2x-0.1\pi). \qquad (4.33)$$

The first term of this function was used in [50]. The second term is added here to make the function even more complicated. Here, a hundred ($P = 100$) uniformly

distributed random samples over [0, 1] were chosen for network training. The signal-to-noise ratio (SNR) is selected to be 10 [dB]. The number of uniformly sampled noiseless data for evaluating the generalization FVU is two hundred (200). The training and the generalization FVUs are shown in Figure 4.3. It is seen clearly that the proposed algorithm works slightly better than the "standard" algorithm in terms of the generalization FVU.

Example II: Consider the following two-dimensional regression function (HF):

$$\begin{aligned} g(x_1, x_2) &= 42.659\ (\ 0.1 + (x_1 - 0.5)(0.05 + (x_1 - 0.5)^4 \\ &\quad -10(x_1 - 0.5)^2(x_2 - 0.5)^2 + 5(x_2 - 0.5)^4)\ ) \end{aligned} \tag{4.34}$$

This harmonic function was used earlier in Chapter 2. Here, two hundred and twenty-five ($P = 225$) uniformly distributed random samples were generated from the interval [0, 1] for network training. The SNR is 10 [dB]. Ten thousand (10,000) uniformly sampled points from the same interval without noise were used for generalization. The simulation results are given in Figure 4.4. It can be observed from this figure that the proposed new algorithm results in considerably smaller generalization FVU than the "standard" algorithm. This suggests that the proposed new algorithm may be more useful for more "complicated" regression problems. Here, we also show the generalized surfaces by both the proposed and the "standard" algorithms for the regression function HF, in Figure 4.5 and Figure 4.6, respectively.

To further verify this statement four other two-dimensional regression functions, as used in [45, 51] and given below, are used for simulations.

- *Additive function (AF):*

$$\begin{aligned} g(x_1, x_2) &= 1.3356\left\{1.5(1 - x_1) + e^{2x_1 - 1}\sin(3\pi(x_1 - 0.6)^2)\right. \\ &\quad \left. + e^{3(x_2 - 0.5)}\sin(4\pi(x_2 - 0.9)^2)\right\}, \end{aligned} \tag{4.35}$$

- *Radial function (RF):*

$$g(x_1, x_2) = 24.234\left\{((x_1 - 0.5)^2 + (x_2 - 0.5)^2)(0.75 - (x_1 - 0.5)^2 - (x_2 - 0.5)^2)\right\}, \quad (4.36)$$

- *Simple interaction function (SIF):*

$$g(x_1, x_2) = 10.391\left\{(x_1 - 0.4)(x_2 - 0.6) + 0.36\right\}, \quad (4.37)$$

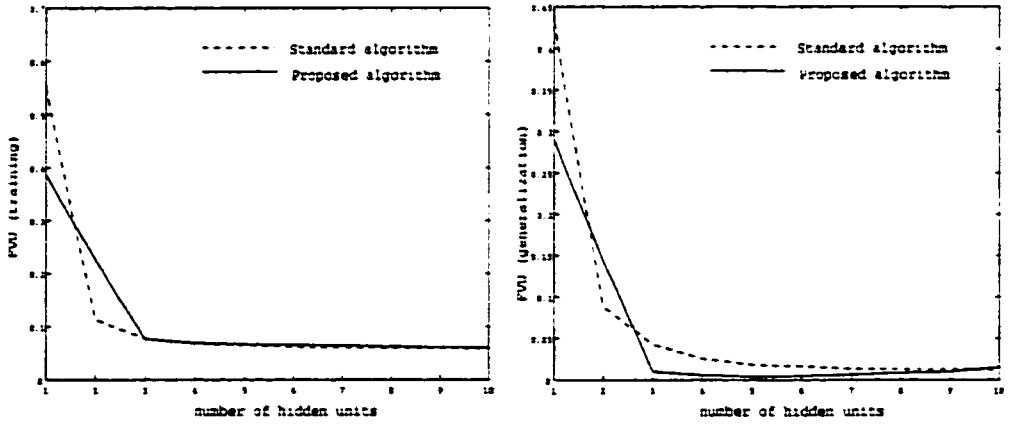
- *Complicated interaction function (CIF):*

$$g(x_1, x_2) = 1.9\left\{1.35 + e^{x_1 - x_2}\sin(13(x_1 - 0.6)^2)\sin(7x_2)\right\}. \quad (4.38)$$

Simulations for the above four functions (SNR=10 [dB]) are performed under the same settings as in the HF case. Simulated results for these five functions are now summarized Table 4.1. Obviously, the generalization FVUs are significantly improved by using our proposed algorithm. For SIF and AF functions both algorithms yield similar training FVUs as the number of hidden units increases, although our proposed algorithm results in smaller training FVUs for the other functions regardless of how large the number of hidden units becomes.

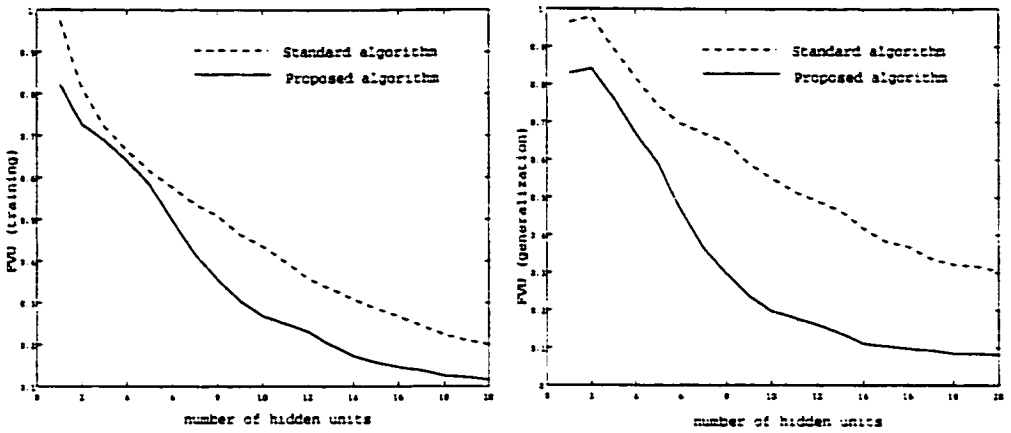
Table 4.1: Mean FVU values for the training and the generalization of our proposed and the "standard" constructive OHL networks for the five two-dimensional regression functions considered in [45] (SNR=10 [dB]).

		Number of hidden units							
		Training				Generalization			
Function	Approach	2	5	10	20	2	5	10	20
CIF	"standard"	0.582	0.345	0.196	0.107	0.620	0.403	0.206	0.111
	proposed	0.556	0.348	0.138	0.087	0.594	0.349	0.095	0.039
AF	"standard"	0.630	0.208	0.132	0.101	0.700	0.156	0.073	0.050
	proposed	0.501	0.218	0.151	0.109	0.538	0.166	0.076	0.027
HF	"standard"	0.812	0.614	0.434	0.201	0.980	0.744	0.551	0.303
	proposed	0.726	0.582	0.268	0.116	0.842	0.590	0.197	0.082
RF	"standard"	0.634	0.224	0.115	0.073	0.668	0.212	0.097	0.048
	proposed	0.441	0.155	0.092	0.074	0.443	0.091	0.027	0.019
SIF	"standard"	0.297	0.153	0.099	0.072	0.298	0.119	0.048	0.043
	proposed	0.334	0.157	0.100	0.079	0.284	0.095	0.031	0.026



(a) Training FVUs (b) Generalization FVUs

Figure 4.3: Structure of a constructive OHL-FNN that takes the orthonormal Hermite polynomials as the activation functions of its hidden units.



(a) Training FVUs (b) Generalization FVUs

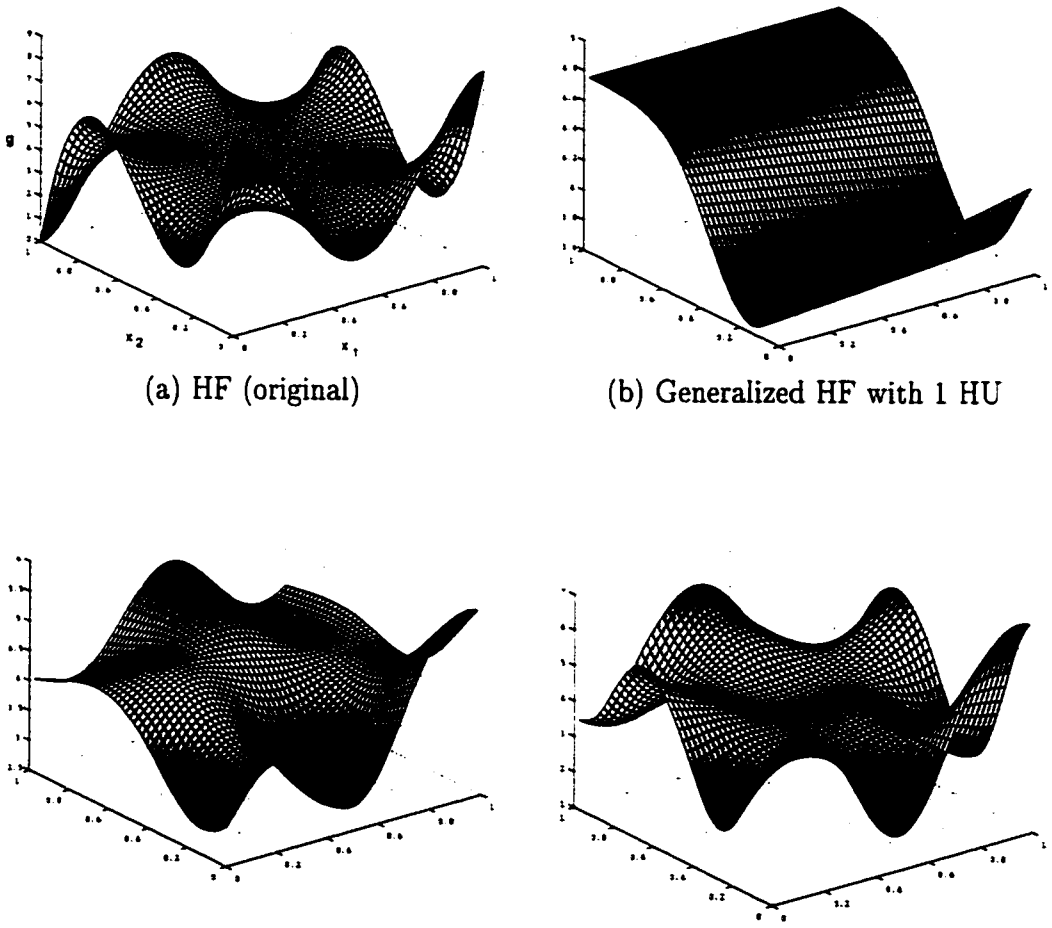
Figure 4.4: (a) Training and (b) generalization FVUs of the new and the standard constructive OHL networks for the HF.

Example III: Consider the following three-dimensional regression function given by

$$g(x_1, x_2, x_3) = \frac{1}{1 + e^{-e^{x_1} + (x_2 - 0.5)^2 + 3\sin(\pi x_3)}} + 2x_1 \sin(2\pi x_2). \tag{4.39}$$

A thousand ($P = 1000$) uniformly distributed random samples within the interval [0, 1] were used for network training. Also, a thousand (1000) different samples were uniformly generated for generalization performance testing. The SNR is chosen to be 10 [dB]. The results for this function are depicted in Figure 4.7. Obviously, in this case the new proposed algorithm works much better in terms of the generalization *FVU* than the previous standard algorithm.

Using all the above simulated results for the regression problems one can observe that the new constructive network with Hermite polynomials as activation functions learns from the training samples as well as and even better than the previous network with sigmoidal activation functions, and also produces smaller generalization FVUs. In other words the proposed network provides similar or even



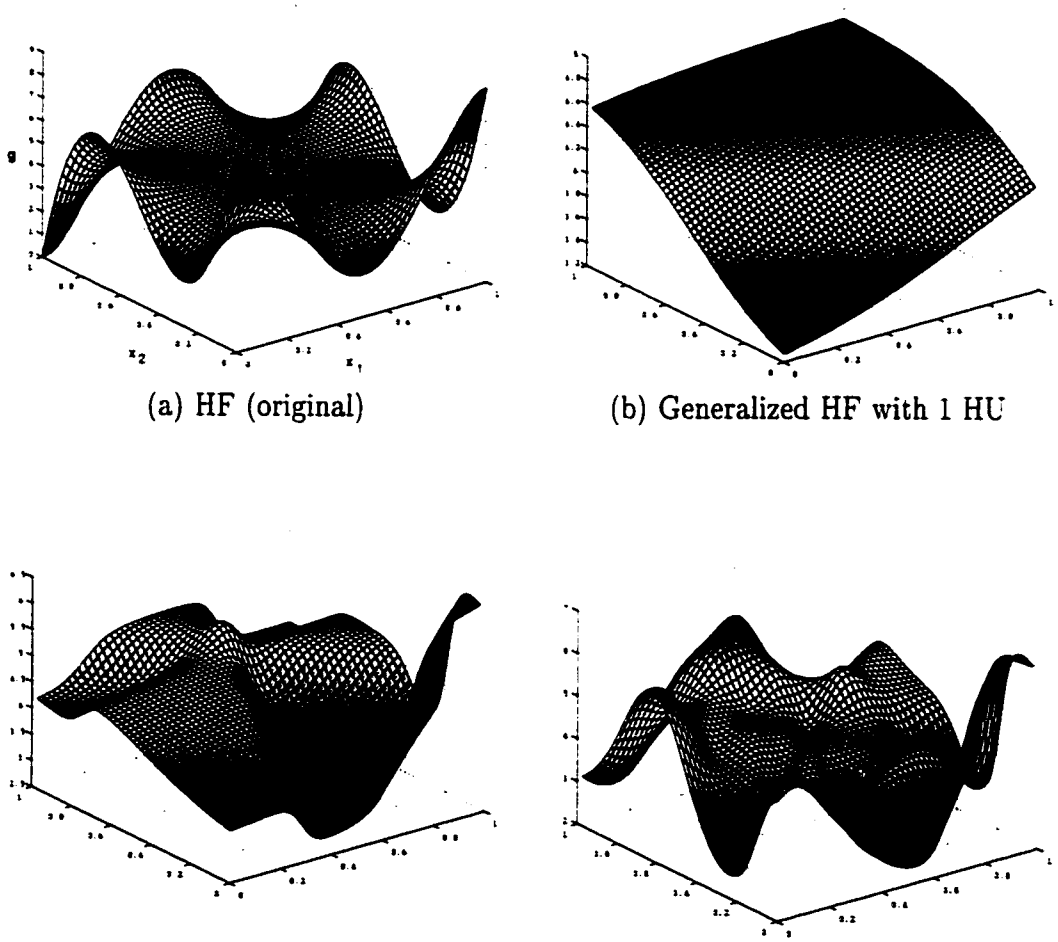
(a) HF (original)

(b) Generalized HF with 1 HU

(c) Generalized HF with 5 HUs

(d) Generalized HF with 20 HUs

Figure 4.5: Original and generalized HF by the proposed algorithm.



(a) HF (original)

(b) Generalized HF with 1 HU

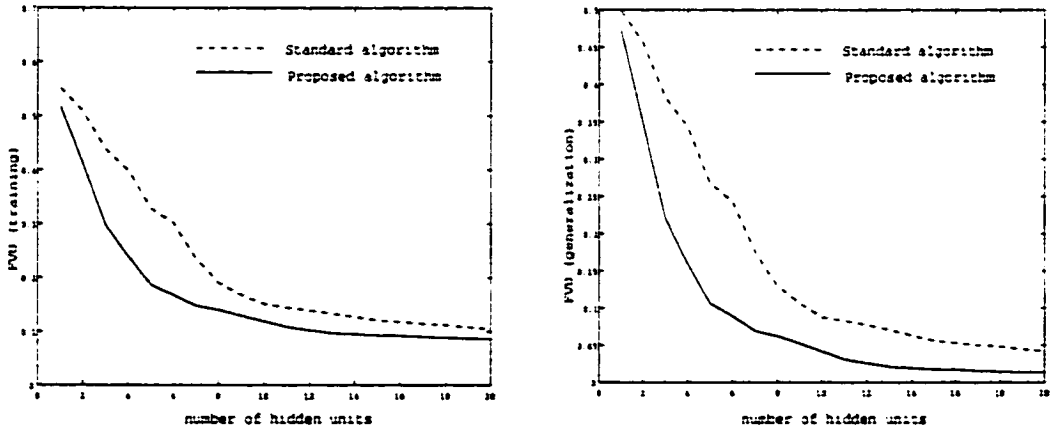
(c) Generalized HF with 5 HUs

(d) Generalized HF with 20 HUs

Figure 4.6: Original and generalized HF by the "standard" algorithm.

improved learning capabilities with the training samples, and is more capable of representing the data and generalizing beyond the training samples.

Finally, to demonstrate that the concepts presented here can equally be applied to classification problems we apply the proposed algorithm to a two-category classification problem. Note that the output node has no longer a linear activation function as it is changed to a sigmoidal function. The output-side training is carried out using the Quasi-Newton algorithm. To make a fair comparison, the training parameters for the previous ("standard") and the proposed algorithms have been assigned such that both algorithms yield approximately their best performances.



(a) Training FVUs (b) Generalization FVUs

Figure 4.7: (a) Training and (b) generalization FVUs of the new and the standard constructive OHL networks for the three-dimensional regression function.

Example IV: Consider the following two-category classification problem:

$$g(x_1, x_2) = \begin{cases} 0 & x_1 < 0,\ 1 \le x_1^2 + x_2^2 \le 4; \\ & or\ \ x_1 > 0,\ x_1^2 + (x_2+1)^2 \le 1. \\ 1 & x_1 < 0,\ x_1^2 + x_2^2 < 1; \\ & or\ \ x_1 > 0,\ 1 < x_1^2 + (x_2+1)^2 \le 4. \end{cases} \tag{4.40}$$

The two categories are sampled at an 0.1 interval in both the horizontal (x_1) and the vertical (x_2) directions to obtain sufficient network training samples. Sampling with the resolution of 0.05 is performed to collect data for network generalization. Figure 4.8 shows the original two categories on a two-dimensional surface. The generalized two categories are given in Figures 4.9 and 4.10, which are obtained from two typical networks of 5 and 10 hidden units trained by both the standard and the proposed constructive algorithms, respectively. Comparisons of the training and the generalization errors (FVUs) for both constructive algorithms are shown in Figure 4.11. From Figures 4.9 - 4.11, it follows very clearly that the proposed constructive FNN using Hermite polynomials as its activation functions yields better performance than the "standard" constructive FNN that uses identical sigmoidal functions as its activation functions.

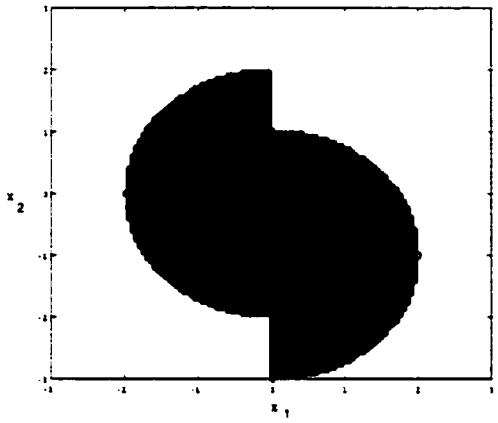
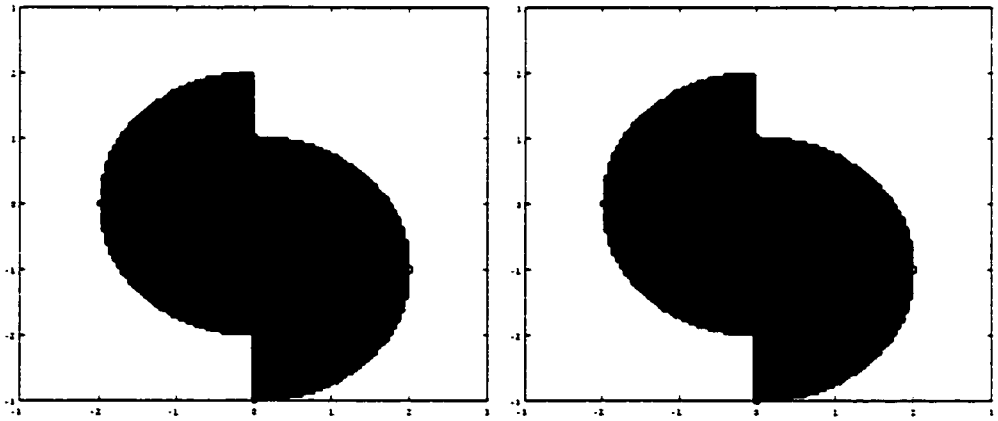
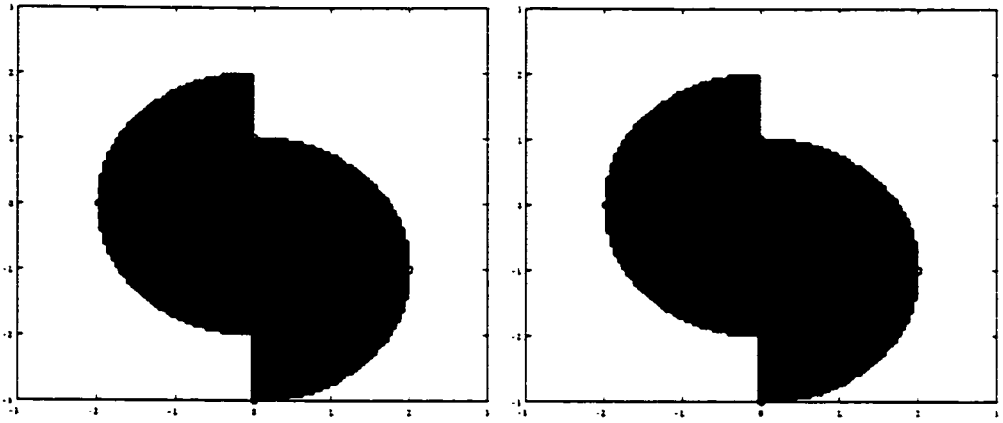


Figure 4.8: The original two categories.



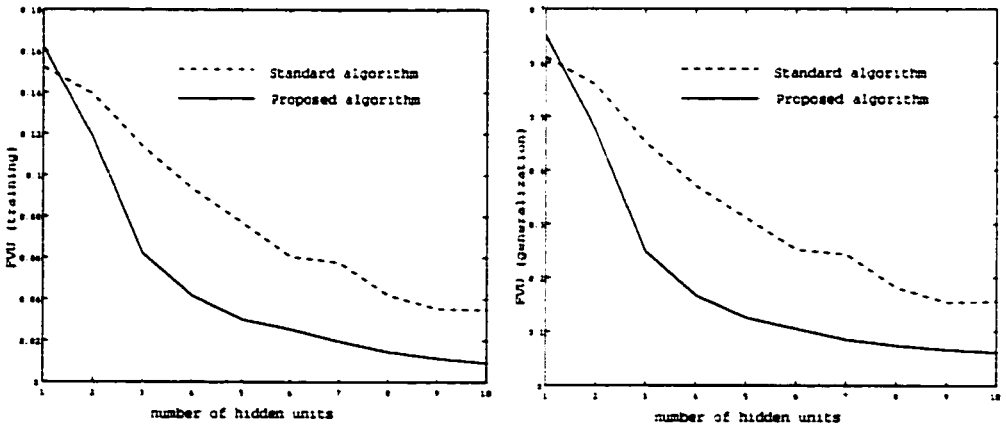
(a) Standard constructive algorithm (b) Proposed algorithm

Figure 4.9: Generalized two categories by the previous and the proposed constructive FNNs with 5 hidden units.



(a) Standard constructive algorithm (b) Proposed algorithm

Figure 4.10: Generalized two categories by the previous and the proposed constructive FNNs with 10 hidden units.



(a) Training FVUs (b) Generalization FVUs

Figure 4.11: (a) Training and (b) generalization FVUs of the new and the standard constructive OHL networks for a two-category classification problem.

4.5 Conclusions

In this chapter, we have proposed a new type of constructive OHL-FNN. This network learns better and as efficiently as, but generalizes much better than the previous ("standard") constructive OHL networks [45]. Extensive simulations for regression problems have been carried out first to confirm the effectiveness and superiority of the proposed new algorithm. Application to an albeit simple classification problem has also been included to demonstrate the potential power of the proposed algorithm. The pruning algorithms proposed in Chapter 2 can also be easily incorporated into the new algorithm when the proposed algorithm is applied to problems of higher dimensional input vector. Furthermore, the constructive policy proposed in Chapter 3 may also be combined with the new algorithm presented in this chapter to construct multi-layer FNNs with polynomial hidden units to represent and learn even more complicated regression problems.

Chapter 5

Applications of the constructive one-hidden-layer FNNs to image compression and facial expression recognition

5.1 Introduction

Digital image presentation requires a large amount of data and its transmission over communication channels is time-consuming. To ease this situation, a huge number of techniques to reduce (compress) the amount of data for representing a digital image have been developed to make its storage and transmission economical. Development of these new techniques is a very important research area within the field of image processing known as image compression.

A vast number of image compression algorithms have been developed in the literature [54]. Particularly, in the past decade numerous attempts have been made to pursue the possibility of using various neural networks (NNs) for image compression (see for example [55, 56, 57] for reviews). Autoassociative neural networks

[58], self-organizing Kohonen map (SOM) [59, 60], cellular neural networks [61, 79], counter-propagation neural networks [62], to just name a few, are among the vast number of techniques that have been proposed in the literature. For example, refer to [63]-[83] for more neural network algorithms that have been used for image compression purposes. Among the large number of neural networks techniques available for image compression, we are particularly interested in Multi-Layer Perceptron (MLP)-type feedforward NNs (FNNs) due to their structural elegance, abundance of training algorithms and good generalization capabilities [71]-[76], [81], [83].

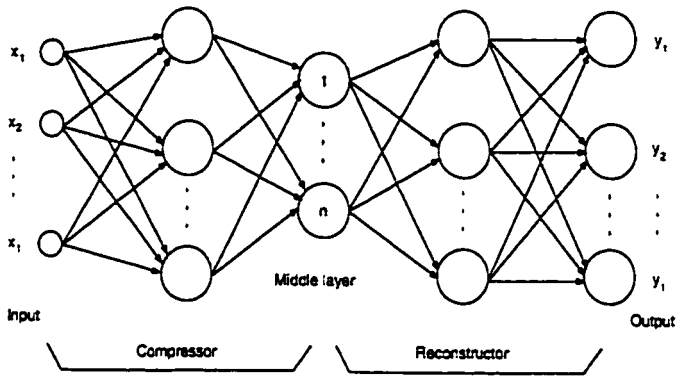
In the above NN-based algorithms, an image is usually divided into small square blocks of pixels which are then framed into patterns for network training. The size of each block is typically taken as 4×4, 5×5, 6×6, or 8×8, or even 16×16, although this generally depends on the nature of the image being compressed and the training algorithm used. The NN considered would then have identical input and output dimensions equal to the number of pixels in a given block.

In a multi-hidden-layer FNN used for image compression [71, 74], the hidden layer in the middle of the network has fewer number of nodes than the input layer, and its output values associated to all the blocks are considered as the compressed image or transmitted image to the receiver. In order to reconstruct the image at the receiver end, the weights for those connections on the right-hand-side (RHS) of the middle layer are also transmitted to the receiver (see Figure 5.1(a) for details). If the number of all the output values for the middle layer units plus their connections on the RHS are less than the total number of pixels of the image, a compression is then achieved. The fewer the number of units in the middle layer, the higher the degree of compression. The multi-hidden-layer FNN considered here is symmetric in structure, with the network from input layer to the middle hidden layer acting as data compressor, while the network from the middle layer to the output layer playing the role of a reconstructor. These two sub-networks have the same number of connections. This technique works quite well as shown by many experimental

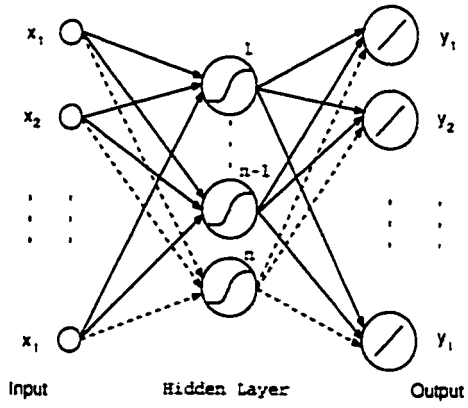
results in the literature [71, 74, 75]. However, there are two basic problems with this strategy. The first one is due to the large network size, which makes the training costly and the training convergence sluggish. The second problem is the determination of a proper network size, in other words generally trial and error seems to be the only available option one has at his/her disposal for choosing the proper compression ratio. The idea of using only OHL-FNN has been proposed in [69, 72] (also see the references therein). This idea may lead to a faster training convergence at the expense of lower quality of the reconstructed image for the same compression ratio. Determination of the number of hidden units is still a open problem that requires costly trial and error runs.

To address the above problem, a constructive OHL network has been proposed for image compression by Setiono and Lu [73] (see Figure 5.1(b)). This algorithm is quite similar to the dynamic node creation algorithm of [35]. It starts with a network that contains only a single hidden unit in the hidden layer. The network is then trained by using a variant of the quasi-Newton method to achieve fast convergence. If the trained network does not satisfy the accuracy requirements, a new hidden unit is added and the whole network is re-trained again. This process will be repeated until a network with the desired accuracy and/or compression ratio is achieved. This compression algorithm is thus flexible in the sense that the user can trade off between image quality and the compression ratio according to some *a priori* specification. However, the algorithm is not as efficient as other constructive algorithms that (i) freeze the input-side weights of the added hidden units to the existing network, and (ii) do not retrain the frozen weights. With the exception of the above work, constructive algorithms have not yet been applied to image compression applications.

In Chapter 2, we have considered a constructive algorithm for a OHL-FNN to produce a network with both possibly improved performance and reduced number of weights. The proposed algorithm is now applied to image compression in this chapter. Specifically, the input-side weight pruning technique is incorporated in the



(a) Multi-hidden-layer FNN for image compression where ideally $n << I$.



(b) OHL-FNN for image compression where ideally $n << I$.

Figure 5.1: FNNs for image compression.

construction of a OHL-FNN. The network training consists of two phases: input-side training phase and output-side training phase. Once a new hidden unit is added to the existing network, its input-side weights will be fixed (frozen) in the training process that follows. Output-side weights of all the hidden units are then updated in the output-side training phase.

In this chapter, we first attempt to compress a still image using our proposed modified constructive OHL-FNN. The image "Girl" is used to train and construct a network, and the image "Lena" is used to test the generalization performance of the resulting constructed network. Influence of the quantization factors on the compressed image is also investigated. Two types of quantization schemes, namely the uniform and the probability density function optimized (pdf-optimized) quantizations are considered. The performance of the constructed network is then compared with the baseline JPEG technique in terms of the PSNRs for the same compression ratios. Furthermore, the generalization ability of the proposed constructive OHL-FNN is investigated in some details in the presence of additive noise during the training and/or generalization of images.

Next, the proposed modified constructive OHL-FNN network is applied to the problem of the compression of moving images (video sequences). Currently several NN-based schemes have been considered in the literature for the compression of moving images, for example, refer to [56], [77]-[80]. However, constructive FNN has not yet been applied for this purpose. Towards this end, we first present three definitions of similarity (correlation) between two images in order to obtain a better understanding of the concepts behind our proposed NN-based video compression techniques. Extensive experimental results are then provided to demonstrate the effectiveness and the potentials of the proposed techniques.

Finally, we also consider here the application of the modified constructive OHL-FNNs to the problem of facial expression recognition. A promising recognition system was reported in [88] for facial expression recognition based on the

two-dimensional discrete cosine transform (2-D DCT) and the BP-type NN. However, determining the network size was handled by trial and error. In this chapter, the constructive OHL-FNN presented previously is applied to solve this problem with surprisingly good results.

The outline of this chapter is as follows. Section 5.2 gives the details of a constructive OHL-FNN for still image compression. Moving image compression using a constructive OHL-FNN is then presented in Section 5.3. Experimental results for facial expression recognition are shown in Section 5.4. Conclusions are included in Section 5.5.

5.2 Still image compression using a constructive OHL-FNN

In this section, we first derive some theoretical results regarding the constructive OHL-FNN that is to be used for compressing still images. Details about network training are omitted here (for details refer to Chapter 2 for description of the input-side and the output-side training, and the input-side pruning of a constructive OHL-FNN). Experimental results using the images Girl and Lena are provided in this section. The influence of the quantization operations on the hidden layer output of the constructed OHL network is also evaluated. Three separate cases namely, (i) no quantization, (ii) uniform quantization, and (iii) pdf-optimized quantization are considered. Approximately 0.1 to 0.5 [dB] PSNR improvement has been observed using the pdf-optimized quantization in comparison with the uniform quantization. It is also confirmed that the pdf-optimized quantization gives almost the same PSNR as that obtained without any quantization operation. Comparison of our proposed technique with the commonly used baseline JPEG is also performed to assess the relative capability and performance of our technique. It is revealed that our proposed technique works as well as or even better than the baseline JPEG. The generalization

capability of our constructed network is investigated in the presence of additive noise during the training and/or generalization of images. Several scenarios are considered for this purpose. It was found that for this application the constructive OHL-FNN is not generally as robust as expected to the additive noise residing in the images.

5.2.1 Constructive OHL-FNN for image compression

In NN-based image compression the image of size $L \times L$ that is being compressed is first divided into square blocks of equal size $M_b \times M_b$. Each square block is then arranged into a vector of dimension $I \times 1$ ($I = M_b^2$) that will be fed to the neural network as an input training pattern. All such vectors are put together to form a training matrix $\mathbf{X}$ of size $I \times P$, where P is the number of square blocks, i.e. $P = L^2/M_b^2$. The target matrix $\mathbf{D}$ for the neural network is considered to be the same as the input matrix $\mathbf{X}$. The schematic of a OHL NN-based image compression is shown in Figure 5.2.

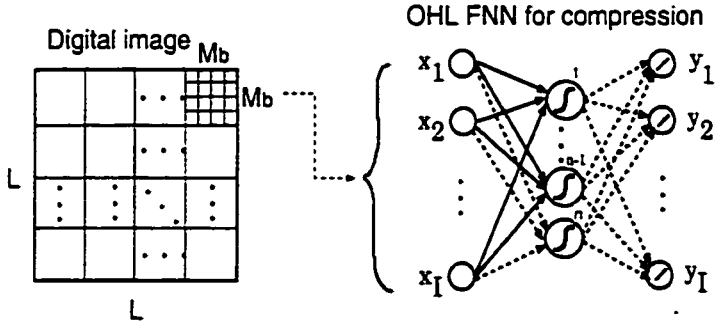


Figure 5.2: A schematic for OHL-FNN-based image compression.

To improve the quality of compression and/or training performance, the input patterns to the neural network are normalized by one of the following rules:

$$x_i^j = U_i^j / U_{max} \tag{5.1}$$

$$x_i^j = 1 - 2U_i^j/U_{max} \tag{5.2}$$

where x_i^j is the normalized i-th element of the j-th input vector, U_i^j is the gray-level value of the original pixel, and U_{max} is the largest possible gray level of the input image being compressed. If one uses (5.1), "logsig" functions can be used as the activation function of the hidden units. The "tansig" functions are selected as the activation functions when one adopts the expression (5.2). In this chapter, we will use the normalization governed by (5.1). The output of the k-th hidden unit for the j-th input pattern is now expressed by

$$z_k^j = f_1\left(\sum_{i=1}^{I} w_{k,i}x_i^j + b_{z,k}\right) \tag{5.3}$$

where $w_{k,i}$, $b_{z,k}$ and f_1 are the input-side weights, bias and activation function of the k-th hidden unit, respectively. As discussed above the activation function f_1, in general, may be selected as a logsig function. The output of the o-th output node will now be given by

$$y_{n-1,o}^j = f_2\left(\sum_{m=1}^{n} v_{y,m}z_m^j + b_{y,o}\right) \tag{5.4}$$

where $v_{y,m}$, $b_{y,o}$ and f_2 are the input-side weights, bias and activation function of the o-th output node, respectively, n is the number of hidden units of the current network, and f_2 can also be a nonlinear activation function, but linear type output activation functions will simplify both network training and network implementation.

The correlation-based objective function for input-side training of the n-th hidden unit is considered as

$$J_{input} = \sum_{o=1}^{I}\left|\sum_{j=1}^{P}(e_{n-1,o}^j - \bar{e}_{n-1,o})(f_1^j - \bar{f}_1)\right| \tag{5.5}$$

where

$$e_{n-1,o}^j = y_{n-1,o}^j - d_o^j \tag{5.6}$$

is the output error at the o-th output node when there are $n-1$ fixed hidden units in the network.

The output-side training will be performed corresponding to the following summed square error criterion

$$J_{output} = \frac{1}{2}\sum_{o=1}^{I}\sum_{j=1}^{P}\left(y_{n,o}^{j} - d_{o}^{j}\right)^{2}. \tag{5.7}$$

5.2.2 Experimental results

First, we provide two definitions for quantifying the measurement of the quality of images reconstructed from the network output. The first is the peak signal-to-noise ratio (PSNR) defined by

$$PSNR \;=\; 10\log_{10}\left(\frac{P\times I\times(255)^{2}}{\sum_{o=1}^{I}\sum_{j=1}^{P}(y_{n,o}^{j}-d_{o}^{j})^{2}}\right), \tag{5.8}$$

and at times the global signal-to-noise ratio (SNR) defined by

$$SNR \;=\; 10\log_{10}\left(\frac{Target\ image\ variance}{\frac{1}{P\times I}\sum_{o=1}^{I}\sum_{j=1}^{P}(y_{n,o}^{j}-d_{o}^{j})^{2}}\right) \tag{5.9}$$

may be used.

The second is the compression ratio ρ that is calculated according to

$$\rho \;=\; \frac{L\times L}{n\times P + n\times I + I + 1} \tag{5.10}$$

where the first term in the denominator is the number of the outputs of the hidden layer with respect to all the input patterns, the second and the third terms are the number of the output side weights and biases, and the final term 1 is the overhead (U_{max}) due to the image normalization. Note that in (5.10) it is assumed that all the above-mentioned numbers are assigned with the same number of bits.

There are two possible approaches that one may adopt to compress an image. The first approach is to train a network corresponding to each single image that will be compressed and then transmitted. This approach is clearly simple in its

compression policy. Network training is performed for each new image. However, this may give rise to some serious problems with the training cost when this approach has to be used in real-time applications. The compression ratio in this case may be evaluated according to the definition (5.10).

The second approach is to train a network corresponding to a series of images with "similar" statistics at first, and then transmit the output of the hidden layer only for a new presented image. Given that the network for compression task is already trained in advance (off-line), the problems of real-time training and convergence will not be present. However, images that have not been presented to network training might have different degrees of "similarities" to those used in network training. These different degrees of similarities could significantly influence and deteriorate the qualities of the reconstructed images. Moreover, a new image that is significantly "different" from those used during the network training could result in a severe quality degradations. To address this problem, a pre-processing procedure is required to monitor and determine the statistics of the new images for quantifying the degree of similarities between images (refer to Section 5.3 for the definitions of the similarity between two images). For instance, histogram or correlation between the new image and the image that has already been used to train the network can be used. If the correlation between the two images is large, the general idea is that there is no need to retrain the network, if on the other hand the correlation is not "sufficiently" large and not "too" small as well, the network may be retrained without a change in its structure to represent the new data, and finally if the correlation is too small, one then would need to retrain the network completely, i.e. to retrain both the weights and the structure of the network simultaneously. The compression ratio in this case may be evaluated by the following expression

$$\rho = \frac{L \times L}{n \times P + 1} \tag{5.11}$$

Two familiar images, the Girl and the Lena, as shown in Figure 5.3, are used

in our experiments. Input-side weight pruning are also invoked in these experiments. Using the second approach (off-line) for image compression (as described above), the input-side weight pruning will result in a reduction in the number of input-side weights, and therefore reduce the implementation cost. To see the relative performance of the constructive NN-based compression technique, the PSNR of a BP-based network having the same number of hidden units as the evolving constructive network solution is also worked out. It should be noted that the whole network is re-trained each time a new hidden unit is added to the network in the BP-based training.

Two typical results are shown below. The first result is the Girl image that is used for training the network, and the second result is the Lena image which is used to test the generalization capability of the trained network. Since the initial weights influence the resulting network in a significant manner, ten (10) runs have been performed to make a statistical performance evaluation. Both images Girl and Lena are of size 512×512. The block size in this case is selected as 4×4. The number of hidden units is increased from 1 to 10. The PSNR's for the training and the generalization are shown in Figure 5.4, respectively. The number of pruned weights is shown in Figure 5.4(c). Clearly the network trained with weight pruning applied has smaller number of weights compared to the standard networks but it yields almost the same PSNR. Note that it is much easier and more efficient to trade off between the compression ratio and the quality of reconstructed image using the OHL constructive network as compared to the conventional BP-based network. This is due to the fact that the PSNR of the constructive OHL depends closely on the number of hidden units each time a new hidden unit is added to the active network, which allows the user to trade off between the PSNR and the compression ratio as the network training evolves. The reconstructed and the generalized images are shown in Figures 5.5 and 5.6, respectively.

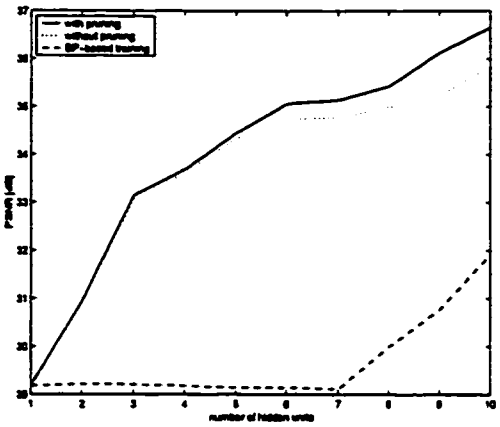
From Figures 5.4(a) and (b), one can empirically specify the relationship between the number of hidden units and the PSNRs of reconstructed and generalized images. The above relationship can be arranged into an expression relating the bit/pixel and the PSNRs by calculating the bit/pixel corresponding to the number of hidden units (n), according to

$$bit/pixel \approx R\frac{n \times P}{L \times L}$$

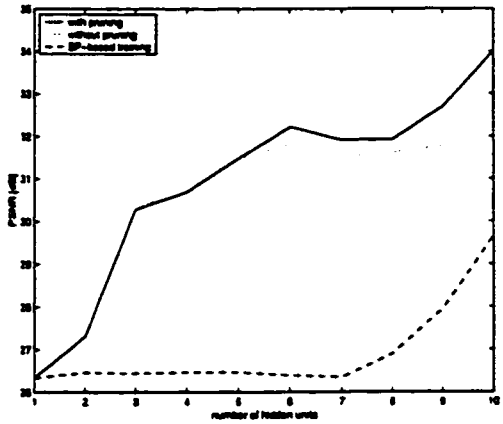
where $R(= 8\ bits/pixel)$ is the bit rate of the original image. In other words, by integrating the results from Figures 5.4 and the above equation, one can simply obtain the plot of PSNRs as a function of the bit/pixel.

(a) The Girl (size 512 × 512) (b) The Lena (size 512 × 512)

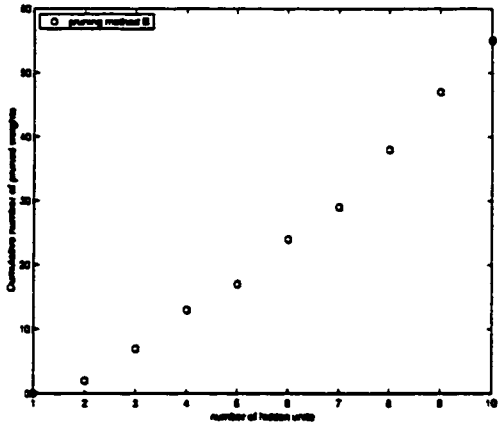
Figure 5.3: Two original images used in the experiments.



(a) Training PSNRs (The Girl image)

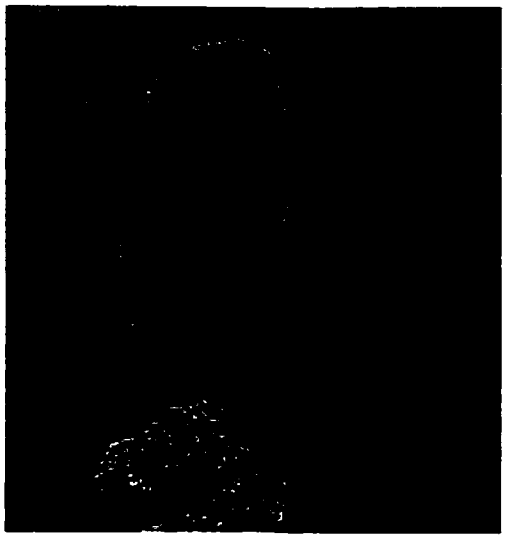


(b) Generalization PSNRs (The Lena image)

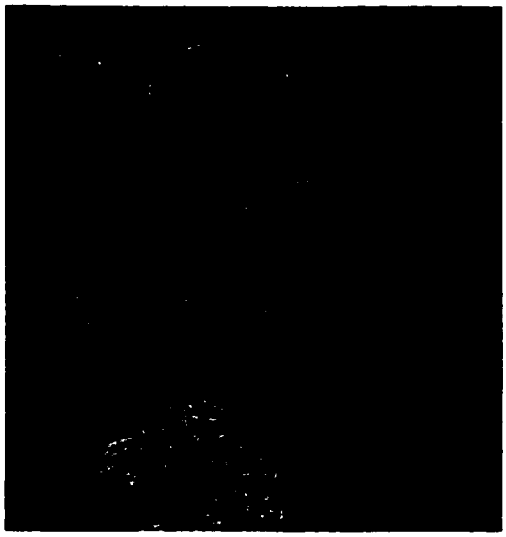


(c) Cumulative number of pruned weights

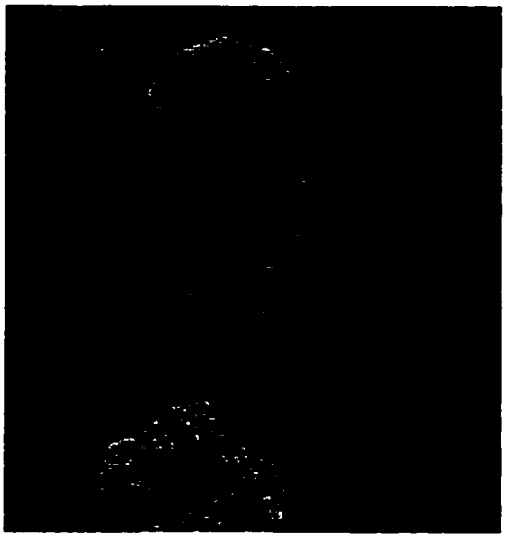
Figure 5.4: PSNRs for (a) training and (b) generalization, (c) the cumulative number of pruned weights.



(a) Constructive training with pruning

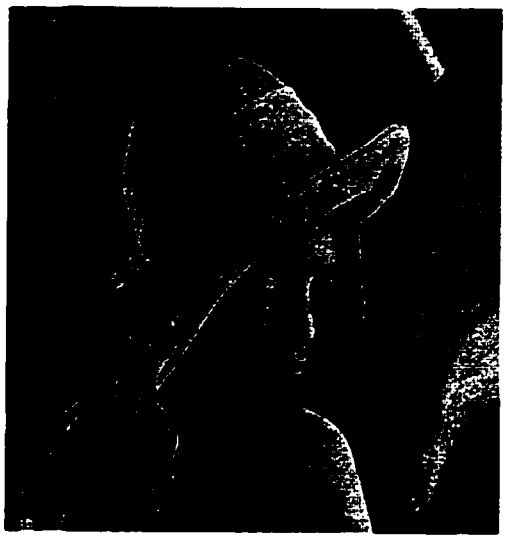


(b) Constructive training without pruning

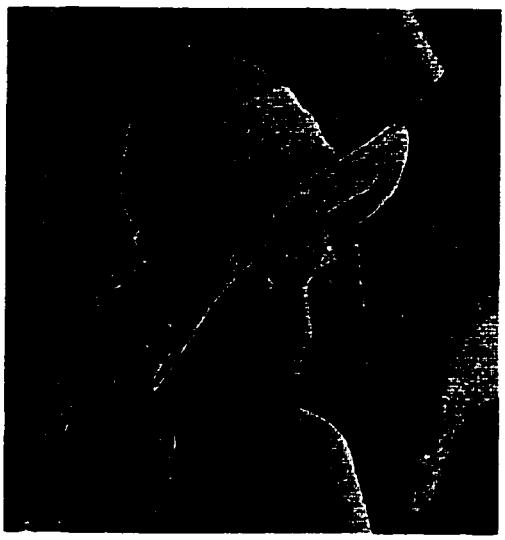


(c) BP-based network training

Figure 5.5: Reconstructed images of the Girl achieved by three networks with 3 hidden units trained by the three respective approaches.



(a) Constructive training with pruning



(b) Constructive training without pruning

(c) BP-based network training

Figure 5.6: Generalized images of the Lena by a network of 3 hidden units trained by the Girl image and the three corresponding training approaches.

5.2.3 Influence of quantization effects

In the proposed technique, the amplitude-continuous output of the hidden layer for a trained or generalized image is to be transmitted in digital form. This requires quantization. The bite rate (R) here is 8 bits/pixel or sample. Let x_k and y_k, $k = 1, 2, \cdots, L_q$, denote the decision levels/values and representation levels/values of a quantizer, respectively, where $L_q = 2^R$ is the number of quantization steps. For a uniform quantizer, we have

$$\begin{aligned} y_{k+1} - y_k &= \Delta; \quad k = 1, 2, \cdots, L_q - 1 \\ x_{k+1} - x_k &= \Delta; \quad for\ finite\ x_k, x_{k+1} \end{aligned} \tag{5.12}$$

where Δ is known as the quantization step size. The actual mapping between x and y represented as

$$y = Q(x) \tag{5.13}$$

is known as the quantizer characteristics. The quantization error is also defined as

$$q = x - Q(x). \tag{5.14}$$

The characteristic $Q(x)$ can be of midtread or midrise type, depending on whether zero is one of the output levels or not. If the data to be quantized represents a uniform probability distribution, the uniform quantizer will be an optimum selection in terms of the quantization error variance. On the other hand if this is not the case, a non-uniform quantizer has to be used.

In this section a typical non-uniform quantizer, such as the pdf-optimized quantizer [84] is considered for comparison purposes. Non-uniform quantization can be achieved by first compressing the signal x using a non-uniform compressor characteristic $c(\cdot)$, then quantizing the compressed signal $c(x)$ employing a uniform quantizer, and finally expanding the quantized signal using $c^{-1}(\cdot)$ which is the inverse of the compressor $c(\cdot)$. This is known as the companding technique (compressing and

expanding). Searching for an optimal compressor characteristic $c(\cdot)$ is the major task in designing a non-uniform quantizer. For high bit rate applications where R and, therefore L_q, are both sufficiently large, an approximate compressor characteristic $c(\cdot)$ can be derived for the pdf-optimized nonuniform quantizer as

$$c_{opt}(x) = x_{max}\left[\int_0^x \sqrt[3]{p_x(x)}dx\right]\left[\int_0^{x_{max}} \sqrt[3]{p_x(x)}dx\right]^{-1} \tag{5.15}$$

where $p_x(\cdot)$ is the pdf of the data to be quantized, and x_{max} is the upper bound of x. Given $c_{opt}(x)$ uniform decision levels $kx_{max}/L_q; k = 1, 2, \cdots, L_q$, the corresponding optimum decision level $x = x_{k,opt}$ can be obtained using the above expression.

In our experiments, the output of the hidden units are quantized, where $R = 8, L_q = 256$, and $x_{max} = 1$. The Girl image is used to train the network, while the Lena image is used for network generalization. The patch size is 4×4 as before. Below, we show the pdfs of the hidden layer outputs for the constructed networks having different number of hidden units for the Girl image (Figure 5.7). Clearly, as the number of hidden units increases, the pdf behaves more abnormal (in other words there is more deviation from the uniform probability distribution). Interestingly, opposite to one's expectation, these pdfs all show non-uniform and non-Gaussian distribution forms.

Three cases are now considered: no quantization, uniform quantization, and pdf-optimized quantization. The PSNRs of the three cases for the trained Girl image are evaluated and are compared and plotted in Figure 5.8. It is clear that the pdf-optimized quantization results in almost the same PSNR as that of the no quantization case, implying that the pdf-optimized quantization makes little, if any, PSNR degradation. The pdf-optimized quantization works better than the uniform quantization due to the non-uniform pdf property of the data. The performance advantage of the pdf-optimized over the uniform quantization in this experiment is about 0.1 [dB] at the earlier stages of network training, and is increased to around 0.5 [dB] as the network grows with more hidden units. This is due to the fact that the pdf of the data becomes more deviated from the uniform pdf as more hidden

units are added to the network and more data needs to be quantized.

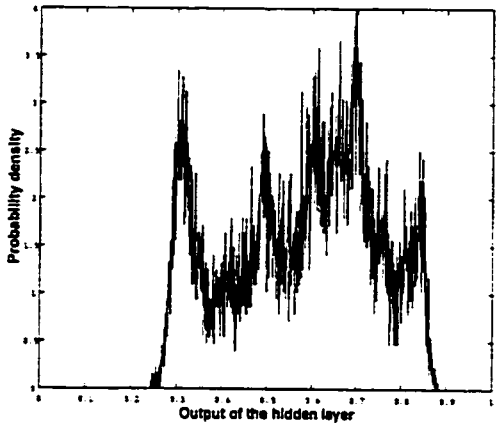
From Figure 5.8 one can also observe that the difference between the PSNRs obtained without quantization and by the uniform quantizer increases as the number of hidden units increases. However, the PSNR of the pdf-optimized quantizer stay close to the PSNR obtained without quantization regardless of the number of hidden units. Therefore, one may conclude here that the uniform quantizer behaves more abnormally as more hidden units are added to the network. Furthermore, the pdf-optimized quantizer behaves excellently even when the number of hidden units increases and the pdf becomes more abnormal.

It should be noted that PSNR variance of 0.5 [dB] over the entire image results in an almost undetectable difference for a human vision system. When putting the proposed NN-based technique into real applications, the uniform quantizer may be chosen if one considers both its simplicity and its acceptable PSNR variance in comparison with other methods.

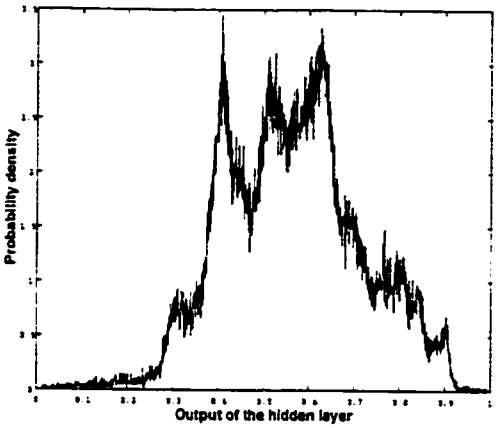
5.2.4 Comparison with the baseline JPEG

In this section the proposed constructive compression technique is compared with the most commonly used still picture compressor JPEG. Comparison is made in terms of the PSNRs of the reconstructed images as a function of the compression ratio.

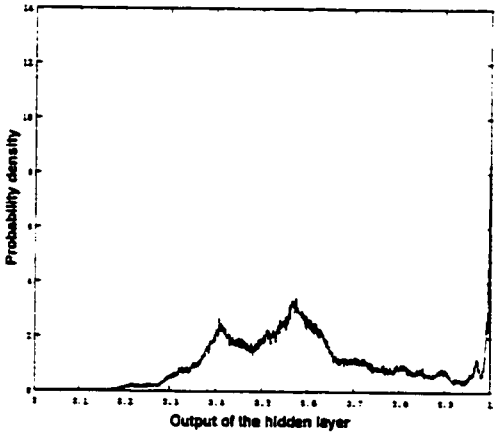
The Girl image is first used to train the constructive OHL-FNN. Reconstructed images for different number of hidden units (in effect different compression ratios) are saved for subsequent PSNR evaluation during the constructive training process. The reconstructed images are obtained by quantizing the output of the hidden layer by using both the uniform and the pdf-optimized quantization schemes. To make a fair comparison with the JPEG algorithm, the compression rate of the quantized hidden-layer output by using the PSNR-lossless Huffman coding is incorporated into



(a) One hidden unit



(b) 5 hidden units



(c) 10 hidden units

Figure 5.7: pdfs of hidden layer outputs for the constructive OHL network with different number of hidden units.

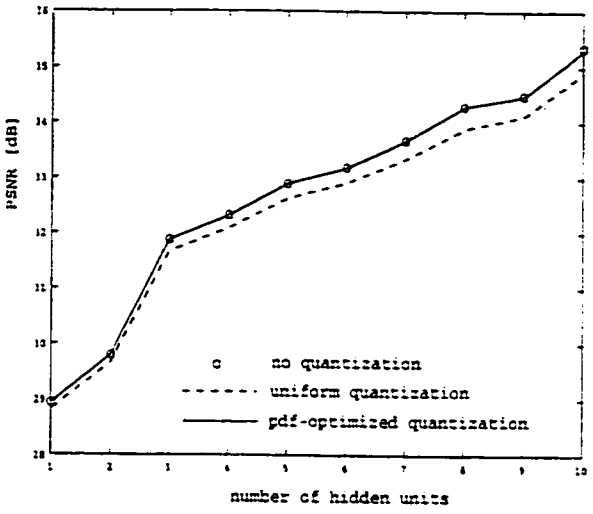


Figure 5.8: PSNRs of the trained Girl image for the three quantization settings.

the total compression ratio for the trained constructive OHL-FNN.

The baseline JPEG images are constructed using a free image tool for similar compression ratios obtained by the constructive OHL-FNN. The PSNRs of these images are evaluated by simply comparing them with the original image.

The PSNRs associated with the JPEG scheme and our technique are plotted in Figures 5.9 and 5.10 for the networks with and without input-side weight pruning, respectively, as a function of the compression ratio. It is seen that our technique yields higher PSNRs for both "low" and "hight" compression ratios, and comparable PSNR for "moderate" compression ratio, when compared to the JPEG scheme. It can also be concluded that our technique works as well as or even better than the JPEG, not considering the significant generalization capabilities that our constructive OHL-FNN possesses. Furthermore, it appears that the uniform quantization scheme is more useful than the pdf-optimized quantization scheme as far as the overall performance of our proposed technique is concerned. This implies that combination of uniform quantization and Huffman coding schemes performs

better than the combination of pdf-optimized quantization and the Huffman coding schemes. In addition, the former combination does also require less computational resources.

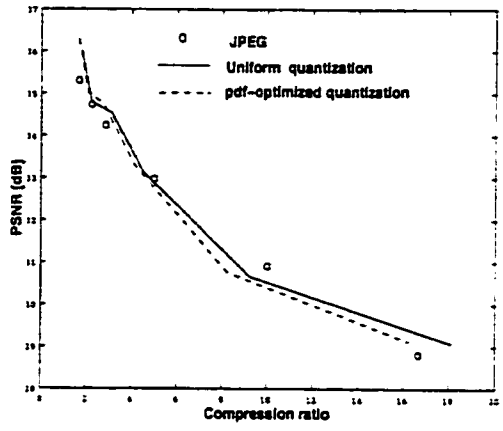


Figure 5.9: Comparison of the PSNRs of reconstructed Girl images obtained by our proposed technique with input-side weight pruning and the JPEG scheme.

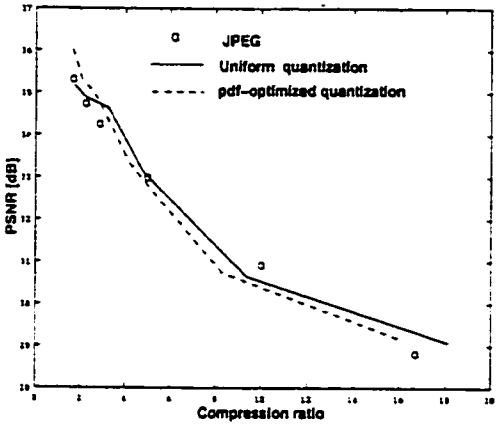


Figure 5.10: Comparison of the PSNRs of reconstructed Girl images obtained by our proposed technique without input-side weight pruning and the JPEG scheme.

5.2.5 Generalization capability of the constructive OHL-FNNs

The constructive OHL-FNNs are found to be generally quite cost-efficient and to produce reasonable performance results as demonstrated in previous chapters of this thesis and in the literature. However, their generalization capabilities in the context of image compression has not yet been investigated systematically. In this subsection we discuss our experimental results regarding this issue and provide some useful and insightful observations. The benchmark images used in our experiments are still the Lena and the Girl images. The Girl image is used to train the network, while the

Lena image is used to test the generalization ability of the trained network. Noisy images are constructed by adding Gaussian white noise to the original clean images. The selected SNR is 10 [dB] for all the images.

In our experiments, the following three scenarios are considered for the constructive network training:

Case T-I: *Noiseless input image and noiseless target image*

This is the case that was considered in the previous Subsection 5.2.2 and where generalization was performed on the noiseless image.

Case T-II: *Noisy input image and noisy target image*

This case may be viewed to more relevant in practice, since images are generally corrupted by noise to a certain degree.

Case T-III: *Noisy input image and noiseless target image*

This case is considered to simultaneously assess the constructed network for its compression performance and noise reduction capabilities.

For each of the above training cases, the following three corresponding generalization cases are considered:

Case G-I *Noiseless input image and noiseless target image*

Case G-II *Noisy input image and noisy target image*

Case G-III *Noisy input image and noiseless target image*

Figures 5.11, 5.12 and 5.13 depict the PSNRs of the reconstructed Girl image subject to the above three network training cases, as well as the PSNRs of the generalized Lena image obtained from the trained network and subject to the above three generalization cases. The following comments and remarks are now in order:

C1 In all the three training cases the PSNRs of the reconstructed images always improve each time a new hidden unit is added to the existing network. This is one of the interesting properties of the constructive OHL-FNN.

C2 In all the three training cases the PSNRs of the generalized images corresponding to noiseless input improve as new hidden units are added to the existing network. This suggests that the constructive OHL-FNNs trained by either noiseless or noisy images generalize well to the noiseless images.

C3 In Case T-I, the noiseless image is generalized with very good quality (Case G-I), while the generalization PSNR for the noisy image improves little as the increase of the number of hidden units is increased in Case G-II. Furthermore, in Case G-III, with the increase of the number of hidden units, the PSNR decreases unexpectedly, which implies that the network trained using noiseless image is not necessarily robust to additive noise.

C4 The constructive OHL-FNN trained using both noisy input and target images (Case T-II) does not demonstrate any ability in filtering the additive noise of the input image used for generalization.

C5 In Case T-III, the network is trained using noisy input and noiseless target images. Obviously, the constructive OHL-FNN can remove the noise in the input image to some degree, however the increase in the number of hidden units improves rather slightly the reconstructed image PSNR. This implies again that the constructive OHL-FNN is not robust to input additive noise, and pre-processing may be needed to remove the additive noise before the constructive OHL-FNN can be used for image compression.

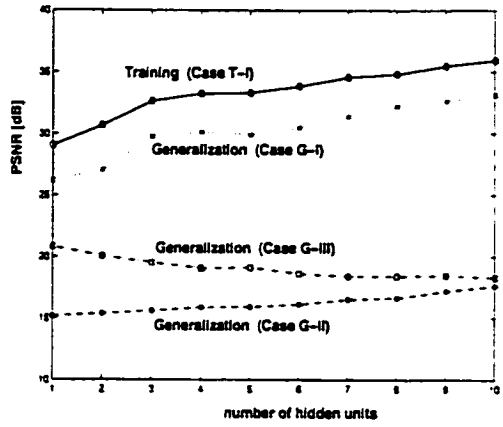


Figure 5.11: The PSNRs of the reconstructed Girl image and generalized Lena image for network training Case T-I and the three generalization cases G-I, G-II, and G-III.

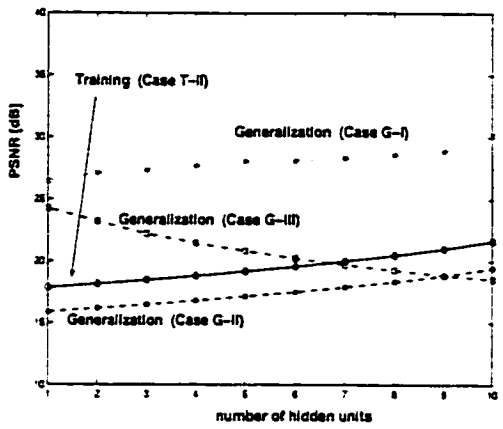


Figure 5.12: The PSNRs of the reconstructed Girl image and generalized Lena image for network training Case T-II and the three generalization cases G-I, G-II, and G-III.

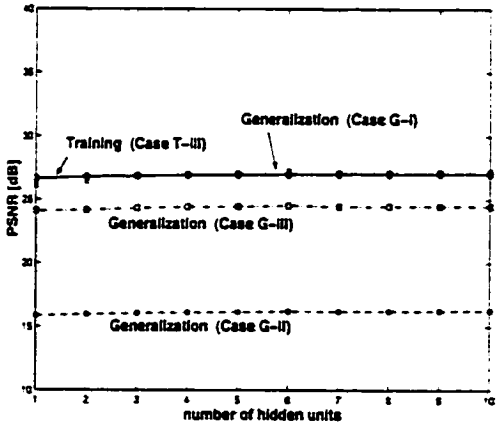


Figure 5.13: The PSNRs of the reconstructed Girl image and generalized Lena image for network training Case T-III and the three generalization cases G-I, G-II, and G-III.

5.3 Moving image compression using a constructive OHL-FNN

A number of techniques have been proposed to date [56], [77]-[80] (also see the references therein) for the use of neural networks for moving image (video sequence) compression. However, to the best of our knowledge, constructive type FNNs for moving image coding have not yet been considered in the literature.

In the previous section, the still image Girl is used to train a constructive OHL-FNN. The resulting trained network is then used to compress the Lena image which was not seen by the network. It was found that the PSNR of the reconstructed Lena image (without noise) is quite high, implying that the proposed technique may be considered for use in certain practical applications. Although, the two images appear visually different, they may actually have some inherent features that are similar on the block basis. This suggests that a certain similarity measure on the block basis may make it possible to use the network that was trained on the Girl image for compressing the Lena image. This motivates one to propose that in moving image compression, one may use a certain number of frames to train a network and then use the resulting trained network to compress some other frames as long as it can be established that latter frames are "similar" to the former images. This has been the key factor in exploring the possibility of using our constructive neural networks for video sequence compression. It should be noted that as long as the similarity between the frame(s) used for FNN training and the subsequent frame is larger than a prespecified value, the latter can be compressed using the trained network, however the temporal redundancy between frames is not exploited here.

5.3.1 Similarity definitions of two images

Suppose a constructive OHL-FNN is trained on a single frame of a moving image sequence, and this network is now being considered to compress the frames that

follow. Naturally, one needs to determine to which degree a given frame is similar to the frame that was used in network training, and how this similarity is related to the compression quality in terms of PSNR (which are all related to the generalization capability of a given network). Thus, it is imperative that a notion of the similarity between two images be clearly defined and quantified. This is clearly of both theoretical and practical importance.

Towards this end, let $\mathbf{Z}_1(i,j)$, $\mathbf{Z}_2(i,j)$, $i = 1 : N_r$, $j = 1 : N_c$ denote two digital images. They are divided into square blocks of size $I_b = I_r \times I_c$. The number of blocks will be $N_b = N_r N_c / I_b$. The blocks are then arranged into vectors $\mathbf{p}_m$ and $\mathbf{q}_m$, $m = 1 : N_b$, respectively. In the following we present three potential similarity measures, and subsequently apply the most appropriate one to our application.

Similarity measure I : This definition is motivated from the standard notion of correlation, and is expressed as follows:

$$S = \frac{\frac{1}{N_r N_c}\sum_{i=1}^{N_r}\sum_{j=1}^{N_c}(\mathbf{Z}_1(i,j) - \mu_1)(\mathbf{Z}_2(i,j) - \mu_2)}{\sqrt{V_1 V_2}} \tag{5.16}$$

where

$$\mu_1 = \frac{1}{N_r N_c}\sum_{i=1}^{N_r}\sum_{j=1}^{N_c}\mathbf{Z}_1(i,j), \tag{5.17}$$

$$\mu_2 = \frac{1}{N_r N_c}\sum_{i=1}^{N_r}\sum_{j=1}^{N_c}\mathbf{Z}_2(i,j), \tag{5.18}$$

$$V_1 = \frac{1}{N_r N_c}\sum_{i=1}^{N_r}\sum_{j=1}^{N_c}(\mathbf{Z}_1(i,j) - \mu_1)^2, \tag{5.19}$$

$$V_2 = \frac{1}{N_r N_c}\sum_{i=1}^{N_r}\sum_{j=1}^{N_c}(\mathbf{Z}_2(i,j) - \mu_2)^2. \tag{5.20}$$

In this definition, the entire image is treated as a stationary stochastic waveform. The quantity S determines the similarity of the two images globally, but is not able to characterize their similarity locally and on the block basis. Although, the definition is simple in form, but quite intensive in calculations.

As an illustration the similarity between the first frame and the 60th frame of a video sequence "Football" (see Figure 5.17 for the original images) is 0.023, and the similarity between the Girl and the Lena images is -0.108. This implies that the above two sets of frames are quite different from each other not only in appearance but also according to this definition. However, it has been shown experimentally that the network trained using the 1st frame of the Football sequence is capable of compressing the 60th frame quite well (see Section 5.3.2 for details). This suggests that the above measure may not be quite appropriate for a NN-based image compression application.

Similarity measure II : In this case the similarity is defined using a block-based average correlation according to

$$S = \frac{1}{N_b} \sum_{m=1}^{N_b} c_m \tag{5.21}$$

where

$$c_m = \max_n \left\{ \frac{\frac{1}{I_b} \sum_{i=1}^{I_b} (\mathbf{p}_n(i) - \mu_{\mathbf{p}_n})(\mathbf{q}_m(i) - \mu_{\mathbf{q}_m})}{\sqrt{V_{p_n} V_{q_m}}} \right\} \tag{5.22}$$

$$\mu_{\mathbf{p}_n} = \frac{1}{I_b} \sum_{i=1}^{I_b} \mathbf{p}_n(i), \tag{5.23}$$

$$\mu_{\mathbf{q}_m} = \frac{1}{I_b} \sum_{i=1}^{I_b} \mathbf{q}_m(i), \tag{5.24}$$

$$V_{\mathbf{p}_n} = \frac{1}{I_b} \sum_{i=1}^{I_b} (\mathbf{p}_n(i) - \mu_{\mathbf{p}_n})^2, \tag{5.25}$$

$$V_{\mathbf{q}_m} = \frac{1}{I_b} \sum_{i=1}^{I_b} (\mathbf{q}_m(i) - \mu_{\mathbf{q}_m})^2. \tag{5.26}$$

Note here that the size of images $\mathbf{Z}_1$ and $\mathbf{Z}_2$ may be different as long as their block size is the same. That is to say, two images with different sizes may appear to be similar on the block basis, and can therefore be compressed by the same NN. For each vector $\mathbf{q}_m$ in image $\mathbf{Z}_2$, a search is performed among all

the vectors of image $\mathbf{Z}_1$ to determine a vector that yields the largest correlation with it. The averaged maximum correlation values of all the vectors in $\mathbf{Z}_2$ is then defined as the similarity between $\mathbf{Z}_1$ and $\mathbf{Z}_2$.

Using this definition, the similarity between the first frame and the 60th frame of the Football video sequence for a block size of 4×4 is 0.914, implying that both images are quite similar on the block basis (note that the similarity between the 60th frame and the first frame may have a different value in this case). The similarity between the Girl and the Lena for the same block size of 4×4 is 0.880. This could explain why the network trained on the 1st frame (block size 4×4) can also generalize or compress the 60th frame with an acceptable PSNR as shown in the next section. Upon closer inspection, one realizes that the computational complexity of this metric due to its intensive search routine, large number of multiplications and sorting operations involved is quite excessive. This is a major drawback of the definition, which has led us to the following modification.

Similarity measure III : This definition is presented to overcome the limitations of the previous metric as follows:

$$S = \frac{1}{N_b} \sum_{m=1}^{N_b} c_m \tag{5.27}$$

where

$$c_m = 1 - \frac{E_{min}(m)}{(2^R - 1) I_b}, \quad R = 8 \; bits/pixel, \tag{5.28}$$

$$E_{min}(m) = \min_n \left\{ \sum_{i=1}^{I_b} |\mathbf{p}_n(i) - \mathbf{q}_m(i)| \right\}. \tag{5.29}$$

According to this definition, the number of required multiplications is substantially reduced as compared to measure II, while the essence of the similarity introduced in measure II has been preserved. Therefore, this measure can be evaluated with much less computational load, and hence may be applied in

practice. The denominator of the RHS of (6.28) may be replaced by $\sum_{i=1}^{I_b} \mathbf{q}_m(i)$ as an alternative normalization factor. Using this measure, the similarity between the first frame and the 60th frame from the Football video sequence is about 0.978, while the similarity between the Girl and the Lena is about 0.983. These values are higher than the similarity values obtained by measure II (the similarities between the 60th frame and the 1st frame, and between the Lena and the Girl may also yield different values).

To summarize, the similarity measure III seems to be the most suitable metric for determining the similarity of two images and has also a much lower complexity as compared to measure II. In video coding, this similarity may be used to detect a scene change, and therefore to automatically restart the network training. Furthermore, the PSNR of a generalized image can not be determined simply by the similarity between the images used for network training and generalization, although the similarity between the two images is expected to play an important role in the PSNR evaluation of the generalized image. This is due to the fact that the structure and connections of the trained network affect to a significant degree this evaluation process. It is not difficult to see that the PSNR of the generalized image may be evaluated by the similarity and the trained network. However, we have not yet found an analytical approach for performing this evaluation.

It should be pointed out that there is an easy and intuitive way to determine how similar a frame used for network training is to a new subsequent frame: Input the new frame to the trained network, and evaluate the PSNR of the network output. The decrease of the PSNR from that of the reconstructed image from training may serve as an indication of the similarity of the two images. This way although simple and direct in nature, however is only a brute force method and tells us *a priori* quantitatively little about the similarity between any two images.

Before presenting the experimental results of our technique for the Football video coding, some more results are provided to show that for the same trained

network, different PSNRs result for different generalized images whose original pictures have different similarities with respect to the image used to train the network. Specifically, the Girl image is used to train a network, but this time the Lena and the "Lake"(512 × 512) are used to check the generalization ability of the resulting trained network. The Lake image is a natural landscape and clearly different from a human picture. The original image of the Lake is given in Figure 5.14. The Lake is of the same size as the Girl and the Lena (512 × 512). The similarities between the Girl and the Lena and the Lake are 0.983 and 0.972, respectively, for a block size of 4×4. The similarities between the Girl and the Lena and the Lake are 0.969 and 0.950, respectively, for a block size of 8×8. It is clear that the larger the block size, the smaller the similarity. The PSNRs for the reconstructed Girl image and the generalized Lena and Lake images are shown in Figures 5.15 and 5.16, respectively. It can be observed from these two figures that having different similarities would result in different generalization PSNRs. Furthermore, the larger the block size, the smaller the similarity and consequently the lower the PSNRs for the reconstructed and generalized images.

5.3.2 Experimental results

The constructive OHL-FNN is now used to compress the a video sequence known as Football. The first frame (see the original image in Figure 5.17) is used to train the network, and the subsequent frames (see examples in Figure 5.17 for the original 20th, 40th, and 60th frames) are then to be compressed by the trained network.

First, the similarities of the Football sequences are determined by invoking measure III, specifically to calculate the similarities between the first frame and the subsequent frames. Since the block size is expected to affect the similarity greatly, a comparison for using different block sizes is also made and the results are shown in Figure 5.18. From this figure it can also be seen that the larger the block size, the smaller the similarity measure. This is consistent with the observation that smaller

Figure 5.14: Original image of the Lake (size 512 × 512).

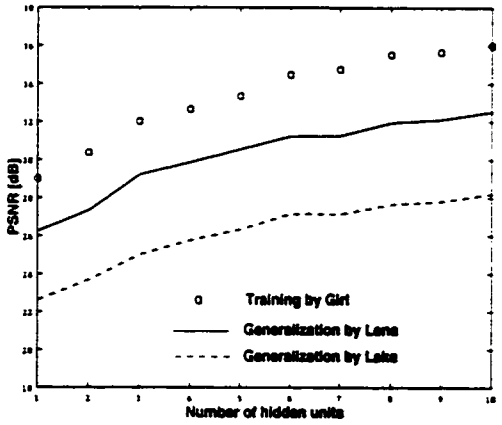


Figure 5.15: The PSNRs of the reconstructed Girl image and the generalized Lena and the Lake images, with the block size of 4 × 4. The similarities between the Girl and the Lena and the Lake are 0.983 and 0.972, respectively.

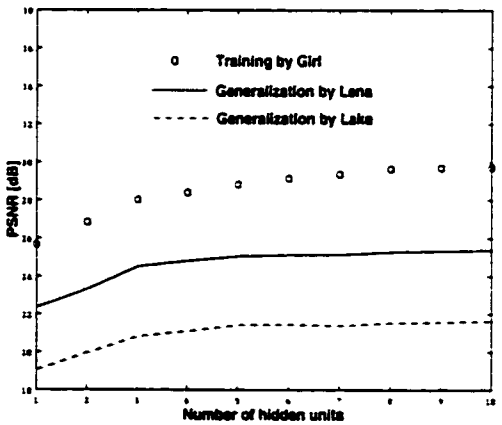


Figure 5.16: The PSNRs of the reconstructed Girl image and the generalized Lena and the Lake images, with the block size of 8 × 8. The similarities between the Girl and the Lena and the Lake are 0.969 and 0.950, respectively.

blocks generally have higher possibility of being similar to each other than larger blocks. It also follows that the similarity, although varying, does indicate small changes within a scene. This implies that the frames within the same scene may be compressed with a similar quality (PSNRs) by the same trained network.

The PSNRs of the reconstructed and the generalized (compressed) images are depicted in Figures 5.19 and 5.20, where the block sizes are selected as 4×4 and 8×8, and the results with and without pruning are presented as a function of the number of hidden units. It can be concluded from these figures that the trained network can successfully compress all the frames that follow the first one. Also, it is shown clearly that the results with pruning is almost the same as those without pruning, where the number of the network input-side weights is reduced by approximately 35% through pruning on the average.

The results for the block size of 16×16 were irregular due to in part the lack of adequate training samples, and are not shown here. The constructive network did not work well for this case as the PSNR of the reconstructed images improved very little or even dropped each time a new hidden unit was added to the network, although the quickprop algorithm for the input-side training converged quite well. Block overlapping technique may be used to alleviate this problem. Furthermore, a larger block size means a larger number of input-side weights, which could make the network training more likely to get trapped in some local minimum. Increasing the number of candidates may lead to some improvement in this case, at the expense of significantly increasing the amount of computational time.

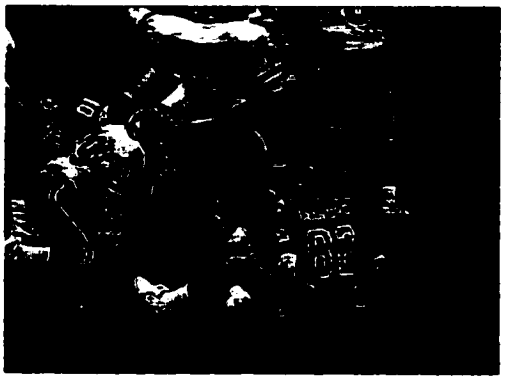
Next in Figure 5.21 we depict four images, namely the reconstructed image of the first frame that was used by the constructive OHL network for training, and the generalized images of the 20th, 40th and 60th frames by the same constructive OHL FNN (see Figure 5.17 for the original images). It is seen that the reconstructed/generalized images are very similar to their original pictures.

Finally, results are now shown to investigate the generalization capability of

(a) 1st frame (original)

(b) 20th frame (original)

(c) 40th frame (original)



(d) 60th frame (original)

Figure 5.17: Original images of the 1st, 20th, 40th and the 60th frames from the Football video sequences (688 × 480 pixels, bite rate R=8 bits/pixel).

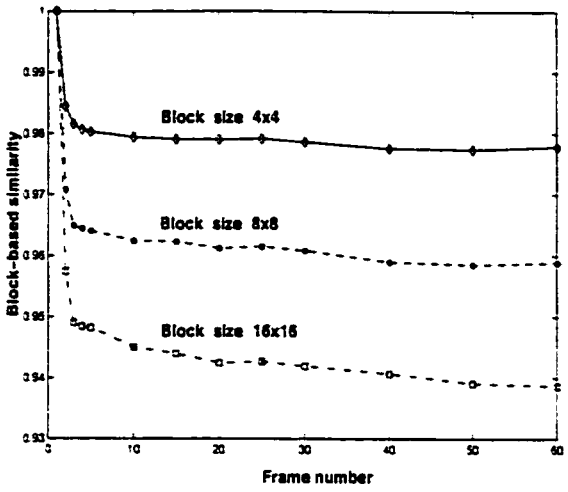


Figure 5.18: Block-based similarity (III) of the subsequent frames with respect to the 1st frame of the Football video sequences, for different block sizes.

the constructive OHL-FNN used for video compression in the presence of additive noise. The first frame is now assumed to be noisy (SNR = 10 [dB]), and is used as the input and target images for network training. The subsequent noiseless frames are now to be generalized by this network. Figure 5.22 shows the PSNRs of the reconstructed or generalized images as a function of the frame number for different number of hidden units. From this figure it follows clearly that (i) the PSNR of the reconstructed image for the first frame improves very slowly as the number of hidden units increases, implying that the constructive training is not effective due to the additive noise in the input and target images, and (ii) the PSNRs of the generalized images for the subsequent frames also improve slowly as the number of hidden units gets larger. Figures 5.23 and 5.24 depict the noisy images for the first frame used for network training and the generalized image for the 20th frame, respectively. Despite the noise in the image for training (Figure 5.23), the trained network can still generalize the 20th frame quite well, although not as good as the

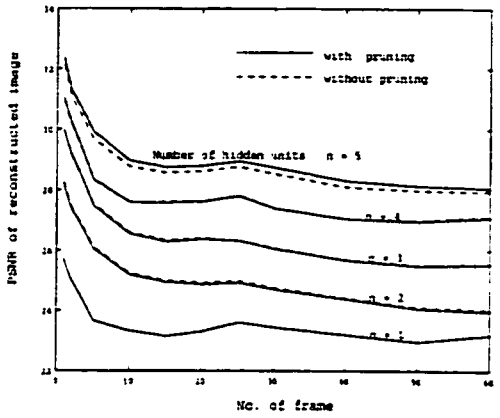


Figure 5.19: The PSNRs of the reconstructed or generalized images vs. the frame number for different number of hidden units, with block size of 4 × 4. The 1st frame was used for network training and other frames were generalized by the trained network. The pruning method B described in Chapter 2 was used with pruning level=10%. The compression ratio is approximately $(16/n) : 1$.

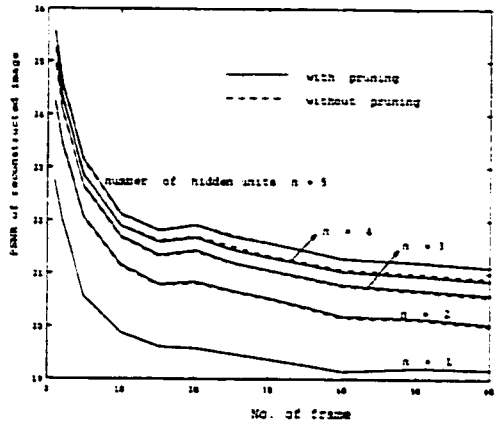
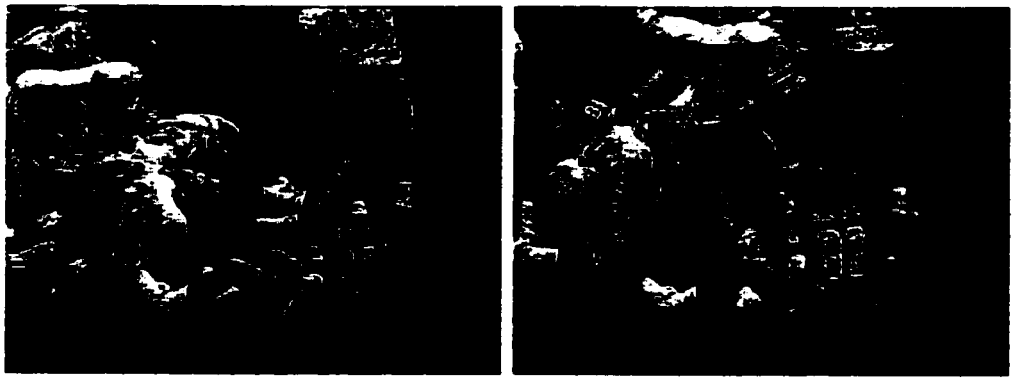


Figure 5.20: The PSNRs of the reconstructed or generalized images vs. the frame number for different number of hidden units, with block size of 8 × 8. The 1st frame was used for network training and other frames were generalized by the trained network. The pruning method B described in Chapter 2 was used with pruning level=10%. The compression ratio is approximately $(64/n) : 1$.

(a) 1st frame (b) 20th frame



(c) 40th frame (d) 60th frame

Figure 5.21: Reconstructed 1st and generalized 20th, 40th, and 60th frame of the Football video sequence. The 1st frame was used to train a constructive OHL-FNN (block size = 4×4, 5 hidden units, network training with pruning method B).

network that was trained by the input image without noise (see Figure 5.21(b) for a comparison). In summary, it can be concluded that the constructive OHL-FNN is not robust to additive noise when used for moving image compression.

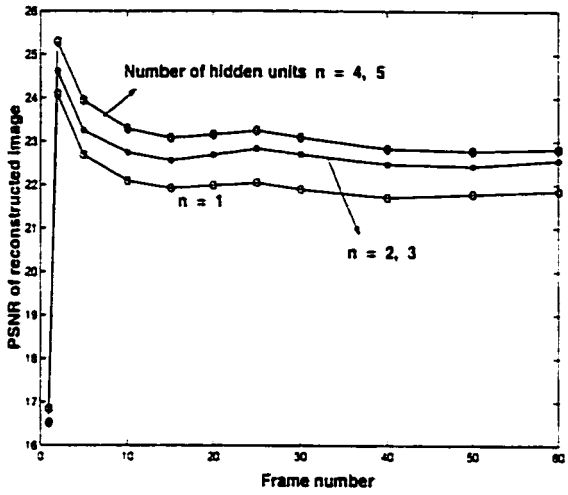


Figure 5.22: The PSNRs of the reconstructed or generalized images vs. the frame number for different number of hidden units, with block size of 4×4. The 1st frame was used for network training and other frames were generalized by the trained network. The pruning method B described in Chapter 2 was used with pruning level=10%, with 38% input-side weights pruned. The compression ratio is approximately $(16/n) : 1$.

Before closing this section, the following comments are in order:

(i) To increase the compression ratio achievable by our technique for video compression, one may use the specific characteristics inherent in the video sequence being considered. Generally speaking, in the same scene, only a number of blocks represent and dominate movement from one frame to another. Therefore, the blocks containing little or no movement in a frame need not to be transmitted. To determine and achieve this, one needs to compare the blocks of the present frame that is being compressed with the corresponding blocks of the previous frame(s) used for network training. The combination with

Figure 5.23: The first frame with additive noise (SNR=10 [dB]) from the Football video sequence, which was used for network training.

Figure 5.24: The 20th frame (Football video sequence) generalized by the network (5 hidden units) trained on the 1st frame with noise.

some proper inter-frame compression techniques, the compression ratio of the constructive OHL may be significantly increased. See [55]-[57] and references therein.

(ii) When the proposed technique is to be used for compression of moving image sequences with successive different scenes of different nature, one may train a series of OHL-FNNs (OHL-FNNs banks) off-line to cope with the different scenes in real-time. This idea is somewhat similar to the well-known vector quantization (VQ) scheme. In this way, for a given scene, the OHL-FNN that captures the best PSNR may be selected and used to compress all the subsequent frames in the scene. Clearly, the comment (i) above can also be integrated into the concept of OHL-FNNs banks.

5.4 Facial expression recognition using a constructive OHL-FNN

5.4.1 Introduction

The computer-based recognition of facial expressions has been an active area of research in the literature for a long time. The ultimate goal in this research area has been the realization of intelligent and transparent communications between human beings and machines. Several facial expression recognition methods have been proposed in the literature, see for example [85]-[88] and the references therein.

A well-known facial action coding system (FACS) was developed by Ekman[85] for facial expression description. In the FACS, the face is divided into 44 action units (AUs), such as nose, mouth, eyes, etc. The movement of muscles of these feature-bearing AUs are used to describe any human facial expression. This method requires 3-dimensional measurement and may thus be too complex for real-time processing. To alleviate the drawbacks with the original FACS, a modified FACS using only 17 relevant AUs was proposed in [86] for facial expression analysis and synthesis. However, 3-dimensional measurement is still needed. The complexity of this modified FACS is reduced when compared to the original FACS, however certain information useful for facial expression recognition may be lost. In recent years facial expression recognition based on 2-dimensional digital images has received a lot of attention by researchers. In [87], a radial basis function (RBF) NN is proposed to recognize human facial expressions. The 2-dimensional discrete cosine transform (2-D DCT) is used to compress the entire face image and the resulting lower-frequency 2-D DCT coefficients are used to train a OHL-FNN in [88]. Very promising experimental results are also reported in [87, 88].

The NN-based recognition methods are found to be particularly promising [87, 88], since the NNs can easily implement the mapping from the feature space of

face images to the facial expression space. However, determining a proper network size has always been a frustrating and time consuming experience for NN developers. This is generally dealt with through long and costly trial-and-error simulations. Motivated by these limitations and drawbacks, in this section we propose to use the constructive FNNs to overcome this problem. The constructive FNNs can systematically determine a proper network size required by the complexity of a given problem, while reducing considerably the computation cost involved in network training when compared with the standard BP-based training techniques. We are particularly interested in the constructive OHL-FNNs considered in Chapter 2, which are simple in structure and present fairly good performances in many applications such as regression problems and image compression [45, 89, 90].

In this section, a new technique for facial expression recognition is proposed which uses the 2-D DCT over the entire face image as a feature detector and the constructive OHL-FNN as a facial expression classifier. An input-side pruning technique proposed in [89] is also incorporated into the constructive learning process to reduce the network size without sacrificing the performance of the resulting network.

This technique is applied to a database consisting of images of 60 men, each having 5 facial expression images (neutral, smile, anger, sadness, and surprise). Images of 40 men are used for network training, and the remaining images of 20 men used for generalization and testing. Confusion matrices calculated in both network training and testing for 4 facial expressions (smile, anger, sadness, and surprise) are used to evaluate the performance of the trained networks. It is demonstrated that the best recognition rates are 100% and 93.75% (without rejection), for the training and testing images, respectively. Furthermore, the input-side weights of the constructed network are reduced by approximately 30% using our pruning method [89, 90]. In comparison with the BP-based recognition method [88], the present technique constructs OHL-FNN with very fewer number of hidden units and weights, while simultaneously providing improved recognition performance.

5.4.2 Application of constructive OHL-FNNs to facial expression recognition

A: Theoretical background

Figure 5.25 describes the procedure in the application of constructive OHL-FNN to the facial expression recognition problem. To recognize the facial expressions from a 2-dimensional human face images, one generally needs to establish a feature detector that can capture the dominant characteristics of the face images, and a classifier that can categorize the facial expressions of interest. The features detected for each facial expression must be insensitive and not be influenced by the appearance of any individual human. Therefore, some preprocessing of the face images is generally needed. One may first obtain a difference image by subtracting a neutral image from a given expression image. The difference images are expected to have much less to do with the appearance of the human whose facial expressions are the subject of recognition. However, it is still very difficult for the classifier to recognize the facial expression from the difference images, as the difference image still has a large amount of data. To facilitate the recognition, one needs to further compress the difference image in order to reduce the size of the data in a proper way, without losing the key attributes and features that play important role in the recognition process. The 2-D DCT is frequently used in image compression as one powerful tool for this purpose. The 2-D DCT can reduce the size of the data significantly by transforming an image into the frequency domain where the lower frequencies possess relatively large amplitudes while the higher frequencies have much smaller magnitudes. That is to say, the higher frequency components can be ignored without significantly compromising the key characteristics of the original difference image, as far as the facial expression recognition problem is concerned. The 2-D DCT coefficients of the lower frequencies capture the most dominant and relevant information of the facial expressions.

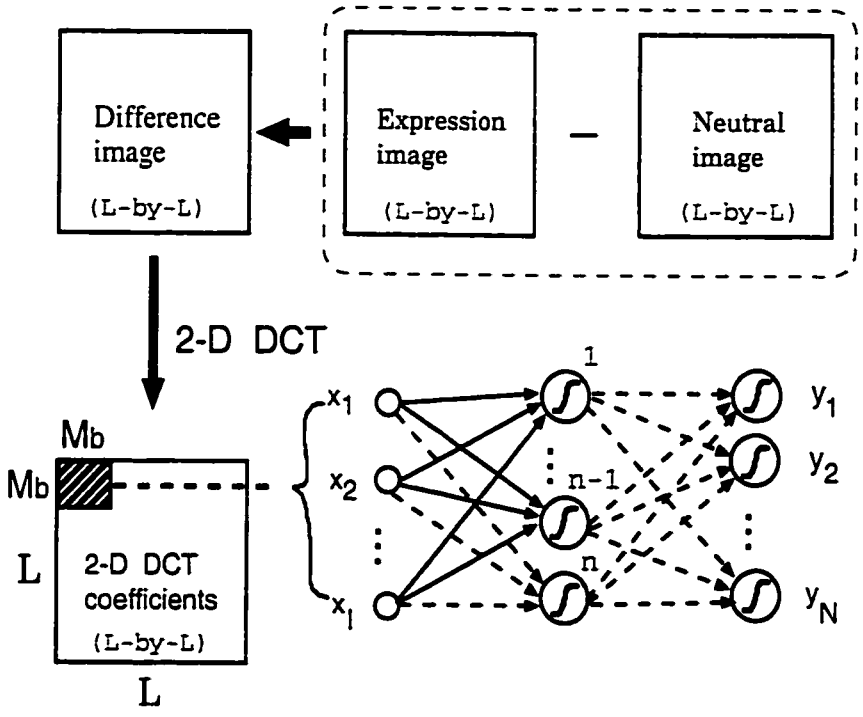


Figure 5.25: Application of the constructive OHL-FNN to facial expression recognition.

The 2-D DCT used here is given by

$$
\begin{aligned}
U_{dct}(k_1,k_2) &= \frac{4C(k_1)C(k_2)}{L^2}\sum_{i=0}^{L-1}\sum_{j=0}^{L-1}\mathbf{Z}(i,j)\cos\frac{(2i+1)k_1\pi}{2L}\cos\frac{(2j+1)k_2\pi}{2L}, \quad (5.30)\\
(k_1,k_2 &= 0,1,\ldots,L-1)\\
C(0) &= \frac{1}{\sqrt{2}},\ \textit{and}\ \ C(k)=1\ \ \textit{for}\ \ k\neq 0.
\end{aligned}
$$

where $\mathbf{Z}(i,j)$ is the gray-level of a pixel of a two-dimensional digital image. A square (or block) of the lower frequency 2-D DCT coefficients ($U_{dct}(k_1,k_2) : k_1 = 0,1,\cdots,(M_b-1), k_2 = 0,1,\cdots,(M_b-1)$) is rearranged as an input vector $\mathbf{x}$ of dimension $I = M_b^2$ to be fed to the constructive OHL-FNN. In this section, 4 facial expressions are considered: smile, anger, sadness and surprise. The target vector is therefore set as $\mathbf{d} = [1\ 0\ 0\ 0]^T$, $[0\ 1\ 0\ 0]^T$, $[0\ 0\ 1\ 0]^T$, *and* $[0\ 0\ 0\ 1]^T$ representing the expressions smile, anger, sadness, and surprise, respectively.

The input-side training is performed by maximizing a correlation objective function given in Chapter 2 based on the *quickprop* algorithm (see also Appendix A for details). Output-side training is performed by using of the Quasi-Newton algorithm due to the nonlinearity of the sigmoidal activation function of the output nodes (see also Appendix B for details). The pruning method B developed in Chapter 2 is used during the network training to reduce the network size. The constructively trained OHL-FNNs are first evaluated by not only the mean summed squared error (SSE) but also their recognition rate subject to these facial expression images that are used during training. The remaining images that are not presented to the network during training will be used to test the generalization capability of the trained networks. Furthermore, the confusion matrix used in pattern recognition is also utilized here to assess the ability of the trained networks to separate the 4 facial expressions being considered. The decision to select a particular facial expression category at the output of the network is achieved by the so-called winner-take-all policy. That is, a given image will be classified to the category whose corresponding output node yields the maximum output value.

B: Experimental results

The constructive OHL-FNN discussed previously in this thesis is applied to a database that consists of images of 60 men, each having 5 face images (neutral, smile, anger, sadness, and surprise). This database is already normalized. In the normalization process, the centers of the eyes and mouth are taken as the reference points, and two lines: one connecting the centers of the eyes (line A), and the other starting from the center of the mouth and ending at the middle point of line A (line B) are considered. An affine transformation is used such that these two lines are orthogonal to each other in all the images. Furthermore, the length of line B is set to a prespecified constant value for all the images. All the images in this database are of size 128×128 having 256 gray levels (bite rate = 8 bits/pixel). Smile, anger, sadness, and surprise are the 4 specific facial expressions of interest. In the simulation experiments, the images of 40 men are used for network training, and the remaining images of 20 men are used for generalization and testing. Figure 5.26 shows a set of samples of face images corresponding to the same man. The facial expression of each face image in this sample set is quite clear to human vision. These face images are to be used in network training. Here, we also give another set of face image samples for another man in Figure 5.27. We see that the facial expression of the fourth image registered as "sadness" is quite difficult even for human to recognize.

Through numerous simulations it was determined that the block size (M_b) of the square of the lower frequency 2-D DCT coefficients used for network training and testing has a strong influence on the NN performance. Therefore, we conducted experiments with different block size M_b. For each M_b, 20 runs with different initial weights were performed to construct 20 OHL-FNNs that have a maximum of 10 hidden units. Their performances were evaluated first by the images used during network training, and then by the remaining images that are not seen by the trained networks.

First, 4 figures (Figures 5.28-5.31) for the mean SSEs of training with pruning,

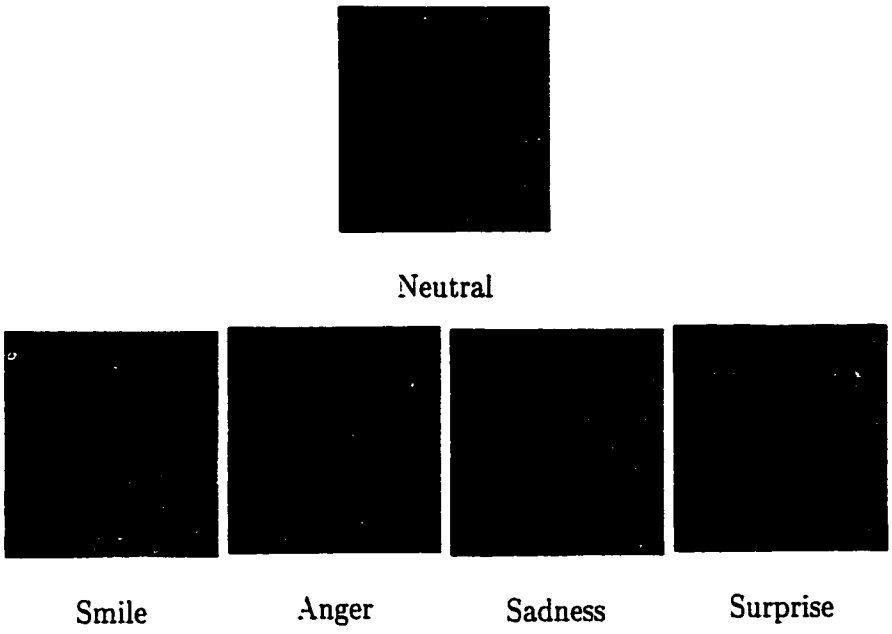


Figure 5.26: Sample of nominal face images from the database.

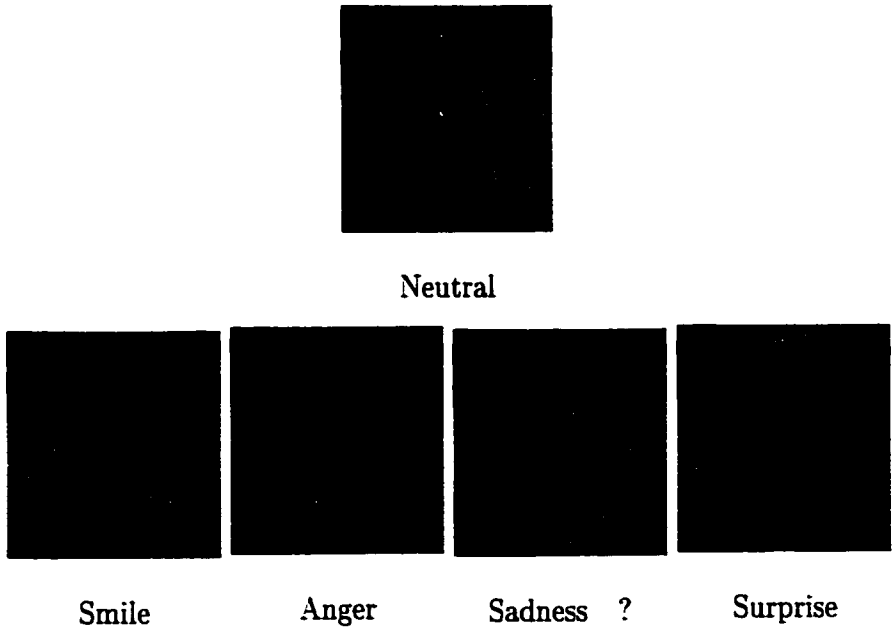


Figure 5.27: Sample of face images from the database, with the image registered as sadness being ambiguous.

the mean SSEs of generalization with pruning, the mean recognition rate in network training, and the mean recognition rate for testing with pruning are presented, respectively as a function of the block size M_b and the number of hidden units. Similar results are also obtained for the 20 OHL-FNNs trained without pruning. Clearly, from Figures 5.28 and 5.30 one can observes that the proposed technique works very well in terms of the training SSEs for all the selected block sizes, and the training saturates when more than 3 hidden units are added to the network. Figures 5.29 and 5.31 indicate that networks with less than 2 and more than 6 hidden units will results in poor performance in terms of both generalization SSEs and recognition rates.

Next, we select the best recognition rates in network training and testing. The purpose here is to decide the most appropriate block size that leads to the highest recognition rates during testing. The results are plotted in Figures 5.32 and 5.33. It is clear that in this database for perfect training the block size needs to be equal to or greater than 8, while for testing the block size of 12 yields the best results. These are based on the 20 runs of network training with or without pruning (40 OHL-FNNs). The best recognition rates obtained during the training and the testing stages with and without pruning are achieved in the 18-th and 8-th runs, respectively. These results will clearly vary if one increases the number of runs.

Finally, we take a closer look at the performance of the constructive OHL-FNN that corresponded to the best block size selected above. In Figures 5.34 and 5.35 we give the mean SSE for training and testing for the block size $M_b = 12$ that resulted in the highest recognition rates in the testing stage. The mean recognition rates during training and testing are also plotted in Figures 5.36 and 5.37, respectively. The mean accumulative number of pruned input-side weights is given in Figure 5.38. The recognition rates with respect to the number of hidden units for the two best OHL-FNNs are provided in Figure 5.39, with one network trained with pruning and the other network trained without pruning. The confusion matrices during training

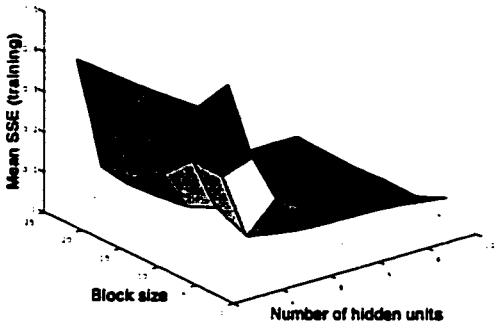


Figure 5.28: Mean training SSEs vs. the block size and the number of hidden units (training with pruning (pruning-level = 0), 20 runs).

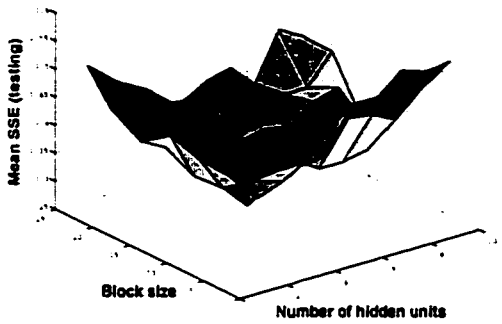


Figure 5.29: Mean generalization SSEs vs. the block size and the number of hidden units (training with pruning (pruning-level = 0), 20 runs).

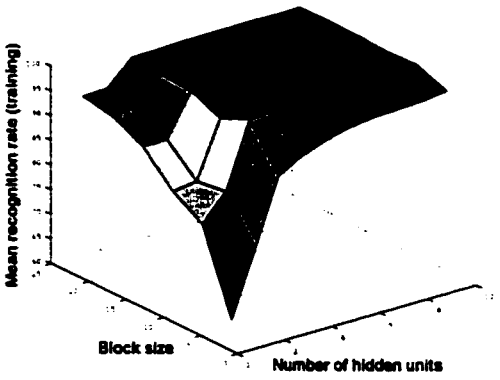


Figure 5.30: Mean recognition rates vs. the block size and the number of hidden units, obtained during network training with pruning (pruning-level = 0) (20 runs).

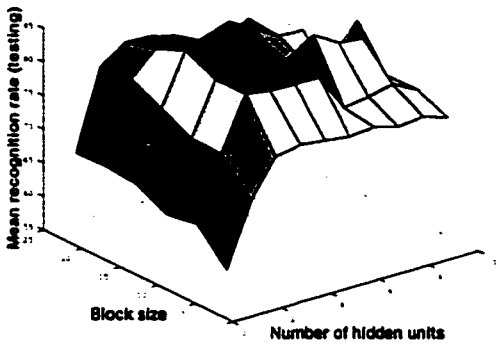


Figure 5.31: Mean recognition rates vs. the block size and the number of hidden units, obtained during testing the networks trained with pruning (pruning-level = 0) (20 runs).

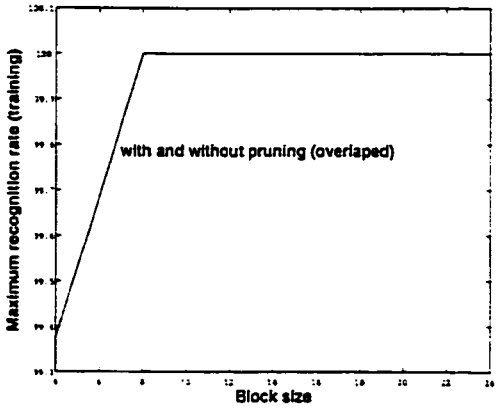


Figure 5.32: Maximum recognition rates vs. the block size obtained during network training with pruning-level = 0 and without pruning (20 runs).

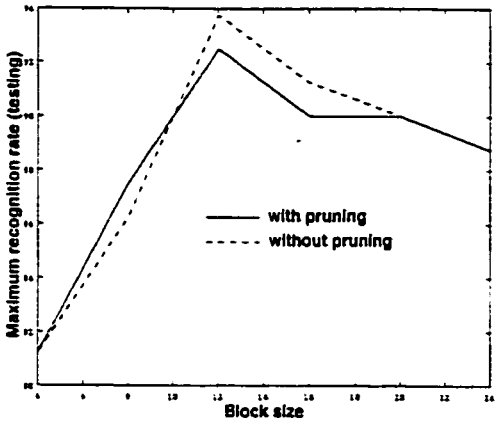


Figure 5.33: Maximum recognition rates vs. the block size obtained in testing for the networks trained with pruning-level = 0 and without pruning (20 runs).

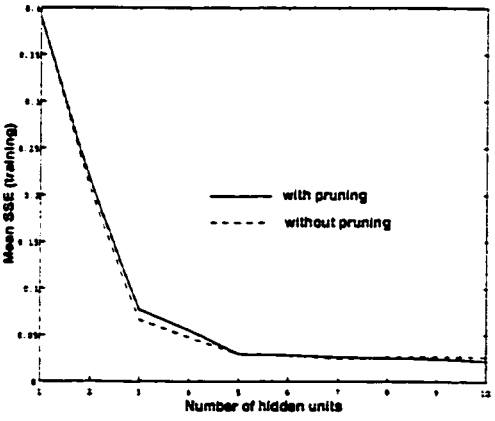


Figure 5.34: Mean SSEs for training of the constructive OHL-FNNs (training with pruning-level = 0 and without pruning, 20 runs).

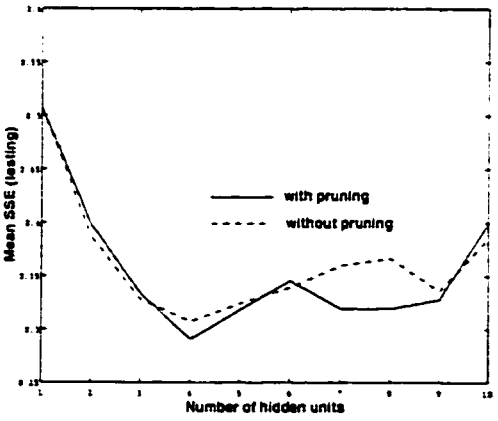


Figure 5.35: Mean SSEs for generalization of the constructive OHL-FNNs (trained with pruning-level = 0 and without pruning, 20 runs).

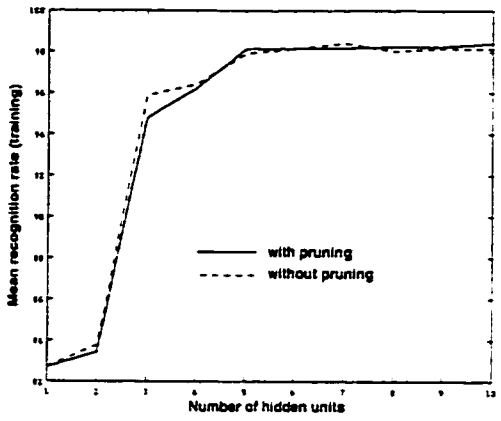


Figure 5.36: Mean recognition rates for the constructive OHL-FNNs obtained during network training with pruning-level = 0 and without pruning (M_b=12, 20 runs).

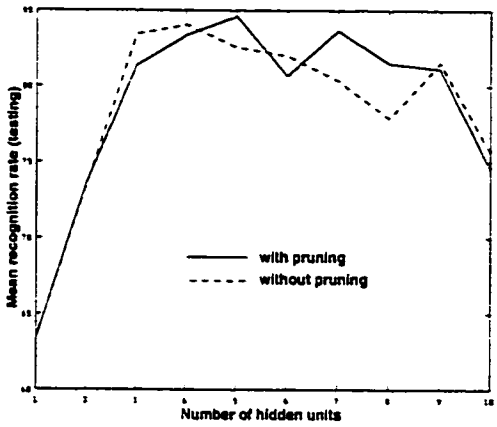


Figure 5.37: Mean recognition rates for the constructive OHL-FNNs obtained for testing of the networks trained with pruning-level = 0 and without pruning (M_b=12, 20 runs).

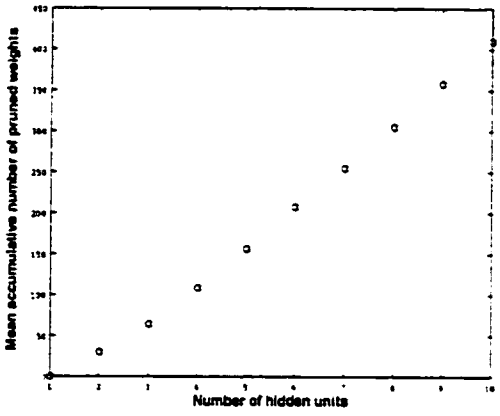


Figure 5.38: Mean accumulative number of pruned input-side weights for the constructive OHL-FNNs with pruning-level = 0, M_b=12 and 20 runs.

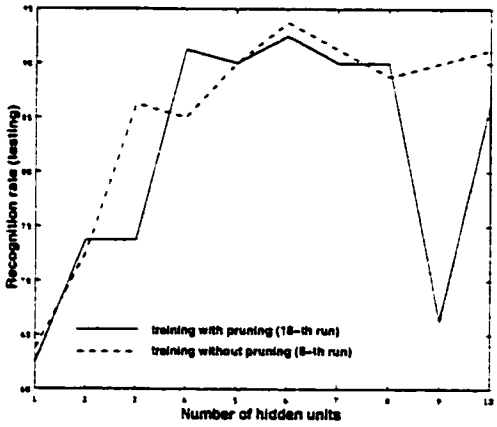


Figure 5.39: Recognition rates vs. the number of hidden units for two constructive OHL-FNNs yielding the best recognition rates in testing stage. These two networks are obtained in the 18-th and 8-th run of network training with and without pruning, respectively (M_b=12).

Table 5.1: Confusion matrix obtained by a OHL-FNN with 6 hidden units trained with pruning (pruning-level=0, M_b=12), for the images used during the network training.

	smile	anger	sadness	surprise
smile	40	0	0	0
anger	0	40	0	0
sadness	0	1	39	0
surprise	0	0	0	40
mean recognition rate 99.375%				

Table 5.2: Confusion matrix obtained by a OHL-FNN with 6 hidden units trained without pruning (M_b=12), for the images used during the network training.

	smile	anger	sadness	surprise
smile	40	0	0	0
anger	0	40	0	0
sadness	0	0	40	0
surprise	0	0	0	40
mean recognition rate 100.00%				

Table 5.3: Confusion matrix obtained by a OHL-FNN with 6 hidden units trained with pruning (pruning-level=0, M_b=12), for the images not seen by the trained network.

	smile	anger	sadness	surprise
smile	20	0	0	0
anger	1	17	2	0
sadness	0	3	17	0
surprise	0	0	0	20
mean recognition rate 92.50 %				

Table 5.4: Confusion matrix obtained by a OHL-FNN with 6 hidden units trained without pruning (M_b=12), for the images not seen by the trained network.

	smile	anger	sadness	surprise
smile	20	0	0	0
anger	0	19	1	0
sadness	1	3	16	0
surprise	0	0	0	20
mean recognition rate 93.75 %				

and testing are given in Tables 5.1-5.4 for these two networks with 6 hidden units.

From the above representative experimental results the following comments are in order:

C1 From Figures 5.28-5.33, it can be noticed that constructive OHL-FNNs trained with or without pruning are found to be capable of learning the training sample images considerably well, and recognizing the new expression images in surprisingly high recognition rates, as long as the block size M_b is properly selected. There may exist an optimal block size for the constructive OHL-FNN which leads to the highest recognition rate during testing. For the database used in this section, the optimal block size is approximately 12. A significantly smaller or a larger block size will result in poor recognition performance.

C2 It can be seen from Figures 5.32-5.37, and 5.39 that the network training with and without pruning results in quite similar performances in terms of the training and generalizing SSEs and recognition rates. However, invoking pruning may reduce the number of input-side weights by approximately 30% resulting in a much smaller network. The OHL-FNNs with 4 to 8 hidden units are found to have sufficient computational capabilities to represent the mapping from the feature space to the facial expression space of face images.

C3 Tables 5.1-5.4 show that the confusion matrices corresponding to expressions "anger" and "sadness" indicate the challenges that the facial expression recognition is faced with when compared to the recognition of expressions "smile" and "surprise".

C4 In comparison with the BP-based recognition algorithm shown in [88], the constructive technique proposed here generates OHL-FNNs with significantly fewer number of hidden units and reduced number of input-side weights while simultaneously resulting in improved recognition performance. A OHL-FNN obtained with block size $M_b = 12$ and a maximum of 6 hidden units yields a

recognition rate that is as high as 93.75% (i.e. 75 expression images are correctly recognized). This result is better than the best result of 94.7% provided by the BP-based networks with block size of $M_b = 16$ and 25 hidden units, and subject to a rejection rate of 5% (see [88]), with actually 72 expression images correctly recognized.

5.5 Conclusions

In this chapter the application of a constructive OHL network to still and moving images compression, and facial expression recognition have been considered.

First, the proposed technique has been applied to the compression of two still images, the Girl and the Lena where promising results were obtained. Influence of quantization effects of the hidden layer output on the network performance was also investigated. It was found that the pdf-optimized quantization scheme works better than the uniform quantization scheme as far as the PSNR of the reconstructed image is concerned. Comparison of the proposed technique with the baseline JPEG has also been made. Experiments have shown that the proposed technique has comparable or even better capabilities as compared to the well-known JPEG scheme. The generalization capability of the constructive OHL network has been investigated in some detail. It was observed that the constructive OHL network is not as robust as expected when used to perform image compression in the presence of additive noise.

Next, the proposed technique was also considered for video sequence compression. Three definitions regarding the similarity between two images were given and clarified. For the same scene, the network trained by the first frame may compress the subsequent frames that follow with quite a good quality of reconstruction provided that the block size is not exceedingly "large".

Finally, the proposed technique has been applied to facial expression recognition. It has been found that the constructive algorithm proposed in Chapter 2 can produce OHL-FNNs with much reduced number of hidden units and input-side weights in comparison with the BP-based NN constructed in [88], while yielding improved recognition rate.

In all the experiments presented, it was revealed that the input-side weight pruning technique proposed in Chapter 2 results in smaller networks, while simultaneously providing similar performance when compared to their fully connected network counterparts.

The constructive OHL network is very attractive from the point of view that the training process is efficient and the user can easily trade off between the PSNR of the reconstructed image and the compression ratio. Further research is needed to expedite the training convergence time at expense of increasing the training computational cost and complexity.

Chapter 6

Conclusions and topics for further research

In this thesis, we first introduced background information on constructive learning, its importance, research progress, and difficulties that are presently limiting this paradigm. We then reviewed several existing constructive learning algorithms such as the dynamic node creation (DNC), activity-based structure level adaptation (ASLA), cascade-correlation (CC), and constructive one-hidden-layer (OHL) networks, presenting their advantages and disadvantages.

Three research directions with new results were then presented. The first direction was the efficient training of hidden units in constructive OHL feedforwad neural networks (OHL-FNNs) [89]. The saturation problem with the correlation-based objective function used in the input-side training was first pointed out, and then a new technique known as error scaling was proposed. This technique can potentially alleviate the saturation problem and increase the training efficiency, and possibly improve the generalization capability and performance of the constructed FNNs. Simulation results revealed that the new technique may help in producing more effective OHL-FNNs. Furthermore, two input-side weight pruning methods were proposed, which can reduce the number of input-side weights significantly

without degrading the generalization performance of the network constructed [89, 90, 96].

The second direction was the development of a new strategy for constructing multi-hidden-layer networks [91, 97]. By using the proposed constructive algorithm, it is feasible to construct a FNN with regular connections and multiple hidden layers. During the constructive learning process, new hidden units and new hidden layers were generated one at a time according to a prespecified criterion when invoked.

The third direction was the use of polynomial activation functions for the hidden units [92, 98]. Each hidden unit has its own polynomial transfer function. All the corresponding activation functions of the hidden units consist of a series of orthonormal polynomials. This idea was basically motivated from the concept of standard series expansion of a given function. The Hermite orthonormal polynomials were used as the activation functions for the hidden units. The FNNs constructed this way works favorably compared to those FNNs that use the identical activation functions such as sigmoidal functions for all of their hidden units [92, 98].

In Chapter 5, the constructive OHL-FNN was applied to still and moving images compression problems [90, 93, 99]. Very promising results were obtained for both still images and video sequences. The results achieved by using the constructive OHL network were compared with the baseline JPEG. The influence of quantization effects and robustness to noise of the constructive OHL networks were also investigated in some detail. Furthermore, the constructive algorithm modified in Chapter 2 was applied to facial expression recognition [94, 95, 100]. Experimental results have shown that OHL network of small size can be easily developed that performs even better than the BP-based networks with larger number of hidden units.

As mentioned above, we have introduced three research directions and presented new results. The experimental experience gained so far will be very instrumental in our future research. Here, we enlist several major research topics that could be possibly explored in future.

T1. Determination of initial weights for new hidden units

In constructive learning of OHL networks, several problems need to be rectified. One problem is the use of candidates, which increases the computation burden to a large extent. To obtain a satisfactory network, sometimes several candidates may need to be considered. All the candidates are started with completely random initial weights. The question to address here may be posed as follows: would it be possible to find a way to reduce the number of candidates? When a new hidden unit is required, it may be feasible to obtain information about its initial weights from the already constructed network. The problem is how to extract this information and formulate it such that it can be directly used in the determination of the initial weights of the candidates. If this problem can be solved, a single candidate would be sufficient. Towards this end, the methodology would be to design an efficient criterion to determine the initial weights of the new hidden units from the already constructed network.

T2. Adaptive order determination of the polynomial activation functions

In Chapter 4, we introduced a constructive OHL network with Hermite polynomial activation functions. The order of the polynomial activation function is increased incrementally by one each time a new hidden unit is included into the network. In other words, the activation functions become more complicated as the network grows. Although the recursive relationships of the Hermite polynomials can be used to reduce the computational complexity of the learning process, the computational burden will become heavy as the network becomes larger. This issue clearly cannot be ignored. A strategy to encourage the use of lower order polynomial activation functions will be more desirable and cost-efficient. Development of such a strategy will also be an attractive topic for further research.

T3. Automatic determination of the user parameters in multi-hidden-layer networks

In Chapter 3, we have introduced a new algorithm for constructing multi-hidden-layer networks. However, the user is required to determine several parameters for the algorithm by trial and error and heuristically. Automatically determining these parameters will be very desirable.

T4. Analysis of the dynamics of the constructive learning algorithms

We have performed no theoretical analysis to reveal the dynamic properties of the constructive learning algorithms. For example, the squared training error as a function of the number of hidden units may be very insightful in understanding the performance limitations of the network and the stability conditions of the training algorithm. It is not difficult to see that theoretical attempt of this nature will be extremely challenging due to the high nonlinearity and dimensionality of the network. This may be one of the most interesting topics to tackle in the future. It seems that a powerful analytical tool has to be first identified and developed before one attempts to analyze the dynamics of the constructive learning algorithms.

Bibliography

[1] R. Lippmann, *"An introduction to computing with neural nets,"* IEEE ASSP Magazine, pp. 4-22, Apr. 1987.

[2] D. R. Hush and B. G. Horne, *"Progress in supervised neural networks,"* IEEE Signal Processing Magazine, pp. 8-39, Jan. 1993.

[3] S. Haykin, *Neural Networks : A Comprehensive Foundation*, Macmillan, 1994.

[4] S. Amari, *"Mathematical foundations of neurocomputing,"* Proc. of the IEEE, vol. 78, No. 9, pp. 1443-1463, Sept. 1990.

[5] N. Kasabov, *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*, Cambridge, Mass, MIT Press, 1996.

[6] J. M. Zurada, *Introduction to Artificial Neural Systems*, West Publishing Company, St. Paul, MN, 1992.

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

[8] B. Mulgerw, *"Applying radial basis functions,"* IEEE Signal Processing Magazine, pp. 50-65, Mar. 1996.

[9] K. S. Narendra and K. Parthasarathy, *"Identification and control of dynamical systems using neural networks,"* IEEE Trans. on Neural Networks, vol. 1, pp. 4-27, Mar. 1990.

[10] K. S. Narendra and K. Parthasarathy, *"Gradient methods for the optimization of dynamical systems containing neural networks,"* IEEE Trans. on Neural Networks, vol. 2, pp. 252-262, Mar. 1991.

[11] S. Haykin, *"Neural networks expand SP's horizons,"* IEEE Signal Processing Magazine, pp. 24-49, Mar. 1996.

[12] T. Chen et al., *"The past, present, and the future of neural networks for signal processing,"* IEEE Signal Processing Magazine, pp. 28-48, Nov. 1997.

[13] I. V. Tetko et al., *"Efficient partition of learning data sets for neural network training,"* Neural Networks, vol. 10, no. 8, pp. 1361-1374, 1997.

[14] D. Sarkar, *"Methods to speed up error back-propagation learning algorithm,"* ACM Computing Surveys, vol. 27, no. 4, pp. 519-542, 1995.

[15] C. T. Leondes, *Neural network systems techniques and applications: Algorithms and Architectures*, Academic Press, 1998.

[16] R. Reed, *"Pruning algorithms — A survey,"* IEEE Trans. on Neural Networks, vol. 4, no. 5, pp. 740-747, 1993.

[17] C. M. Bishop, *Neural networks for pattern recognition*, Oxford:Oxford University Press, 1995.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *"Learning internal representations by error propagation,"* In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, MA, pp. 318-362, 1986.

[19] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis, *"Efficient backpropagation training with variable stepsize,"* Neural Networks, vol. 10, no. 1, pp. 69-82, 1997.

[20] F. Stager and M. Agarwal, *"Three methods to speed up the training of feedforward and feedback perceptrons,"* Neural Networks, vol. 10, no. 8, pp. 1435-1443, 1997.

[21] A. J. Shepherd, *Second-Order Methods for Neural Networks,* Springer-Verlag London Limited, 1997.

[22] S. Osowski, P. Bojarczak, and M. Stodolski, *"Fast second-order learning algorithm for feedforward multilayer neural networks and its applications,"* Neural Networks, vol. 9, no. 9, pp. 1583-1596, 1996.

[23] M. C. Mozer and P. Smolensky, *"Skeletonization: A technique for trimming the fat from a network via relevance assessment,"* in Advances in Neural Information Processing (1), D.S. Touretzky, Ed.(Denver 1988), 1989, pp. 107-115.

[24] E. D. Karnin, *"A simple procedure for pruning back-propagation trained neural networks,"* IEEE Trans. Neural Networks, vol. 1, no. 2, pp. 239-242, 1990.

[25] Y. Le Cun, J. S. Denker, and S. A. Solla, *"Optimal brain damage,"* in Advances in Neural Information Processing (2), D.S. Touretzky, Ed.(Denver 1989), 1990, pp. 598-605.

[26] G. Castellano, A. M. Fanelli, and M. Pelillo, *"An iterative pruning algorithm for feedforward neural networks,"* IEEE Trans. Neural Networks, vol. 8, no. 3, pp. 519-531, 1997.

[27] Y. Chauvin, *"A back-propagation algorithm with optimal use of hidden units,"* in Advances in Neural Information Processing (2), D.S. Touretzky, Ed.(Denver 1989), 1990, pp. 642-649.

[28] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, *"Generalization by weight-elimination applied with application to forecasting,"* in Advances in Neural

Information Processing (3), R. Lippmann, J. Moody, and D.S. Touretzky, Eds., 1991, pp. 875-882.

[29] C. Ji, R. R. Snapp, and D. Psaltis, *"Generalizing smoothness constraints from discrete samples,"* Neural Computation, vol. 2, no. 2, pp. 188-197, 1990.

[30] M. Ishikawa, *"A structural learning algorithm with forgetting of link weights,"* Tech. Rep. TR-90-7, Electrotechnical Lab., Tsukuba-City, Japan, 1990.

[31] W. L. Buntine and A. S. Weigend, *"Bayesian backpropagation,"* Complex Syst., vol. 5, pp. 603-643, 1991.

[32] D. J. C. Mackay, *"Bayesian interpolation,"* Neural Computa., vol. 4, no. 3, pp. 415-447, 1992.

[33] H. H. Thodberg, *"A review of Bayesian neural networks with an application to near infrared spectroscopy,"* IEEE Trans. on Neural Networks, vol. 7, pp. 56-72, 1996.

[34] T. Y. Kwok and D. Y. Yeung, *"Constructive algorithms for structure learning in feedforward neural networks for regression problems,"* IEEE Trans. on Neural Networks, vol. 8, no. 3, pp. 630-645, 1997.

[35] T. Ash, *"Dynamic node creation in backpropagation networks,"* Connection Sci., vol. 1, no. 4, pp. 365-375, 1989.

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

[37] E. B. Bartlett, *"Dynamic node architecture learning: An information theoretic approach,"* Neural Networks, vol. 7, no. 1, pp. 129-140, 1994.

[38] R. Setiono and L. C. K. Hui, *"Use of a quasi-Newton method in a feedforward neural network construction algorithm,"* IEEE Trans. on Neural Networks, vol. 6, pp. 273-277, 1995.

[39] M. R. Azimi-Sadjadi, S. Sheedvash, and F. O. Trujillo *"Recursive dynamic node creation in multilayer neural networks,"* IEEE Trans. on Neural Networks, vol. 4, No. 4, pp. 242-256, 1993.

[40] T. C. Lee, *Structure level adaptation for artificial neural networks,* Kluwer Academic Publishers, 1991.

[41] W. Weng and K. Khorasani, *"An adaptive structural neural network with application to EEG automatic seizure detection,"* Neural Networks, vol. 9, no. 7, pp. 1223-1240, 1996.

[42] S. E. Fahlman and C. Lebiere, *"The cascade-correlation learning architecture,"* Tech. Rep., CMU-CS-90-100, Carnegie Mellon University, 1991.

[43] D. S. Phatak and I. Koren, *"Connectivity and performance tradeoffs in cascade correlation learning architecture,"* IEEE Trans. on Neural Networks, vol. 5, pp. 930-935 (1994).

[44] L. Prechelt, *"Investigation of the CasCor family of learning algorithms,"* Neural Networks, vol. 10, no. 5, pp. 885-896, 1997.

[45] T. Y. Kwok and D. Y. Yeung, *"Objective functions for training new hidden units in constructive neural networks,"* IEEE Trans. on Neural Networks, vol. 8, no. 5, pp. 1131-1148, 1997.

[46] T. Y. Kwok and D. Y. Yeung, *"Bayesian regularization in constructive neural networks,"* Proc. Int. Conf. Artificial Neural Networks, Bochum, Germany, 1996, pp. 557-562.

[47] W. Fang and R. C. Lacher, *"Network complexity and learning efficiency of constructive learning algorithms,"* Proc. Int. Conf. Artificial Neural Networks, 1994, pp. 366-369.

[48] T. M. Nabhan and A. Y. Zomaya, *"Toward generating neural network structures for function approximation,"* Neural Networks, vol. 7, no. 1, pp. 89-99, 1994.

[49] A. I. Rasiah, R. Togneri, and Y. Attikiouzel, *" Modeling 1-D signals using Hermite basis functions,"* IEE Proc.-Vis. Image Signal Process., vol. 144, no. 6, pp. 345-354, 1997.

[50] T. Draelos and D. Hush, *" A constructive neural network algorithm for function approximation,"* IEEE Intl. Conf. on Neural Network, Washington D. C., 1996, vol. 1, pp. 50-55.

[51] J. N. Hwang, et al.. *"Regression modeling in backpropagation and projection pursuit learning,"* IEEE Trans. on Neural Networks, vol. 5, pp. 342-353, 1994.

[52] T. Samad, *"Backpropagation with expected source values,"* Neural Networks, vol. 4, pp. 615-618, 1991.

[53] H. Hashem, *"Optimal linear combinations of neural networks,"* Neural Networks, vol. 10, pp. 599-614, 1997.

[54] A. N. Netravali and B. G. Haskell, *Digital pictures: representation, compression, and standards,* 2nd edition, Plenum Press, 1995.

[55] R. D. Dony and S. Haykin, *"Neural network approaches to image compression,"* Proceedings of the IEEE, 1995, vol. 83, no. 2, pp. 288-303.

[56] C. Cramer, *"Neural networks for image and video compression: a review,"* European Journal of Operational Research, vol. 108, no. 2, pp. 266-282, 1998.

[57] J. Jiang, *"Image compression with neural networks — a survey,"* Signal Processing:Image Communication, vol. 14, no. 9, pp. 737-760, 1999.

[58] A. Basso and M. Kunt, *"Autoassociative neural networks for image compression,"* European Trans. on Telecommunications and Related Technologies, vol. 3, no. 6, pp.593-598, 1992.

[59] O. T. Chen, B. J. Sheu, and W. C. Fang, *"Image compression using self-organization networks,"* IEEE Trans. on Circuits and Systems for Video Technology, vol. 4, no. 5, pp. 480-489, 1994.

[60] C. Amerijckx, M. Verleysen, P. Thissen, and J. -D. Legat, *"Image compression by self-organized Kohonen map,"* IEEE Trans. on Neural Networks, vol. 9, no. 3, pp. 503-507, 1998.

[61] P. L. Venetianter and T. Roska, *" Image compression by cellular neural networks,"* IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications, vol. 45, no. 3, pp. 205-215, 1998.

[62] W. Sygnowski and B. Macukow, *"Counter-propagation neural network for image compression,"* Optical Engineering, vol. 35, no. 8, pp. 2214-2217, 1996.

[63] J. M. Miller and J. N. Peterson, *"An image compression method using a KLT neural network,"* Proceedings of the Artificial Neural Networks in Engineering (ANNIE'95), 1995, vol. 5, pp. 437-443.

[64] E. Gelenbe, H. Bakircioglu, and T. Kocak, *"Image processing with the random neural network,"* Proceedings of the SPIE, 1998, vol. 3307, pp. 38-49.

[65] Q. Wang and Y. Zhong, *"A DCT based image coder with random neural network,"* Mini-Micro Systems, vol. 20, no. 7, pp.481-484, 1999.

[66] A. J. Hussain and P. Liatsis, *"A new recurrent polynomial neural network for predictive image coding,"* Seventh Intl. Conference on Image Processing and Its Applications, 1999, vol. 1, pp. 82-86.

[67] J. S. Lin, *"Image vector quantization using an annealed Hopfield neural network ,"* Optical Engineering, vol. 38, no. 4, pp. 599-605, 1999.

[68] K. I. Diamantaras, K. Hornik, and M. G. Strintzis, *"Optimal linear compression under unreliable representation and robust PCA neural models,"* IEEE Trans. on Neural Networks, vol. 10, no. 5, pp. 1186-1195, 1999.

[69] G. W. Cottrell, P. Munro, and D. Zipser, *"Learning internal representations from gray-level images: An example of extensional programming,"* Proc. 9th Annual Conf. Cognitive Science Society, 1987, pp. 461-473.

[70] L. O. Chua and T. Lin, *"A neural network approach to transform image coding,"* Int. J. Circuit Theory and Applications, vol. 16, pp. 317-324, 1988.

[71] A. Namphol, S. H. Chin, and M. Arozuliah, *"Image compression with a hierarchical neural network,"* IEEE Trans. on Aerospace and Electronic Sys., vol. 32, no. 1, pp. 326-337, 1996.

[72] Y. Benbenisti, D. Kornreich, H. B. Mitchell, and P. A. Schaefer, *"New simple three-layer neural network for image compression,"* Opt. Eng., vol. 36, no. 6, pp. 1814-1817, 1997.

[73] R. Setiono and G. Lu, *"Image compression using a feedforward neural network,"* Proc. IEEE Int. Conf. Neural Networks, 1994, pp. 4761-4765.

[74] G. Panda and P. Singh, *"A novel scheme of data compression and reconstruction using multilayer artificial neural network,"* Journal of the IETE, vol. 41, No.s. 5 & 6, pp. 329-334, 1995.

[75] N. A. Murshed, F. Bortolozzi, and R. Sabourin, *"Binary image compression using identity-mapping backpropagation neural network,"* Proceedings of the SPIE, 1997, vol. 3030, pp. 29-35.

[76] Y. Chang, D. Kumar, and N. Mahalingam, *"Data compression for image recognition,"* Proceedings of the IEEE TENCON - Speech and Image Technologies for Computing and Telecommunications, 1997, pp. 399-402.

[77] C. A. Kamm, G. M. Kuhn, B. Yoon, R. Chellappa, and S.Y. Kung, *"Compressing moving pictures using the APEX neural principal component extractor,"* Proceedings of the 1993 IEEE-SP Workshop, 1993, pp. 321-330.

[78] C. Cramer, E. Gelenbe, and I. Bakircioglu, *"Video compression with random neural networks,"* Proceedings Intl. Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing, 1996, pp. 476-484.

[79] L. Rodriguez, P. J. Zufiria, and J. A. Berzal, *"Video sequence compression via supervised training on cellular neural networks,"* International Journal of Neural Systems, vol. 8, no. 1, pp. 127-135, 1997.

[80] S. Carrato, *"Video data compression using multilayer perceptrons,"* Proceedings of the 9th Italian Workshop on Neural Nets, 1998, pp. 189-194.

[81] A. L. Perrone and G. Basti, *"Novel SAR images compression with de-speckle algorithm,"* Proceedings of the SPIE, 2000, pp. 484-495.

[82] S. Fiori, S. Costa, and P. Burrascano, *"Improved ψ-APEX algorithm for digital image compression,"* Proceedings of the IJCNN, 2000, pp. 392-397.

[83] A. Meyer-Baese, *"Medical image compression by "neural-gas" network and principal component analysis,"* Proceedings of the IJCNN, 2000, pp. 1-5.

[84] N. S. Jayant and P. Noll, *"Digital coding of waveforms: principles and applications to speech and video,"* PRENTICE-HALL, INC., 1984.

[85] P. Ekman and W. Friesen, *Facial Action Coding System,* Consulting Psychologists Press, 1977.

[86] F. Kawakami, H. Yamada, S. Morishima, and H. Harashima, *"Construction and Psychological Evaluation of 3-D Emotion Space,"* Biomedical Fuzzy and Human Sciences, vol. 1, no. 1, pp. 33–42, 1995.

[87] M. Rosenblum, Y. Yacoob, and L. S. Davis, *"Human expression recognition from motion using a radial basis function network architecture,"* IEEE Trans. on Neural Networks, vol. 7, no. 5, pp. 1121-1138, 1996.

[88] Y. Xiao, N. P. Chandrasiri, Y. Tadokoro, and M. Oda, *"Recognition of facial expressions using 2-D DCT and neural network,"* Electronics and Communications in Japan, Part 3, vol. 82, no. 7, pp. 1-11, 1999.

[89] L. Ma and K. Khorasani, *"Input-side training in constructive neural networks based on error scaling and pruning,"* Proceedings of the IJCNN, 2000, vol. VI, pp. 455-460.

[90] L. Ma and K. Khorasani, *"New pruning techniques for constructive neural networks with application to image compression,"* Proceedings of SPIE, 2000, vol. 4055, pp. 298-308.

[91] L. Ma and K. Khorasani, *"A new strategy for adaptively constructing multilayer feed-forward neural networks,"* Proceedings of SPIE, 2000, vol. 4055, pp. 70-78.

[92] L. Ma and K. Khorasani, *"Adaptive structure feed-forward neural networks using polynomial activation functions,"* Proceedings of SPIE, 2000, vol. 4055, pp. 120-129.

[93] L. Ma and K. Khorasani, *"Moving image compression and generalization capability of constructive neural networks,"* Proceedings of SPIE (To appear, 2001).

[94] L. Ma and K. Khorasani, *"Facial expression recognition using constructive neural networks,"* Proceedings of SPIE, vol.4390, pp.187-198, 2001.

[95] L. Ma and K. Khorasani, *"Constructive Hermite polynomial feedforward neural networks with application to facial expression recognition,"* in the Convergence of Information Technologies and Communications (ITCOM 2001), Proceedings of SPIE (To appear).

[96] L. Ma and K. Khorasani, *"New training strategies for constructive neural network with application to regression problems,"* submitted to IEEE Trans. on Neural networks.

[97] L. Ma and K. Khorasani, *"A new strategy for adaptively constructing multilayer feedforward neural networks,"* submitted to Neural Computing.

[98] L. Ma and K. Khorasani, *"Constructive feedforward neural networks using Hermite polynomial activation functions,"* submitted to IEEE Trans. on Neural Networks.

[99] L. Ma and K. Khorasani, *"Application of adaptive constructive neural networks to still and moving image compression,"* submitted to IEEE Trans. on Neural Networks.

[100] L. Ma and K. Khorasani, *"Facial expression recognition using constructive feedforward neural networks,"* submitted to IEEE Trans. on System, Man, and Cybernetics.

Appendix A

The "quickprop" algorithm

The *quickprop* algorithm developed by Fahlman [42] has played a very important role in the input-side training of constructive OHL-FNNs, the multi-layer FNNs, and the polynomial OHL-FNNs (see Chapters 2, 3, and 4 for details). 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 loosely on Newton's method. As shown below, it is simple in form, but has been found to be surprisingly effective when used iteratively [42].

The correlation-based objective function for input-side training is reproduced here (see Eqs. (2.10), (4.26)):

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

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

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

$$s_n^j = \sum_{i=0}^{M} w_{n,i} x_i^j \tag{A.4}$$

where it is assumed that there are already $n-1$ hidden units in the network, and the above objective function is used to train the input-side weight $\{w\}$ of the n-th hidden unit, and I is the number of output nodes, $f_n(\cdot)$ is the activation function of the n-th hidden unit, x_i^j is the i-th element of the input vector of dimension $M \times 1$ $e_{n-1,o}^j$ is the output error at the o-th output node, and $f_n(s_n^j)$ is the output of the n-th hidden unit, all defined for the training sample j. The derivative of J_{input} with respect to an input-side weight is given by

$$\frac{\partial J_{input}}{\partial w_{n,i}} = \sum_{j=1}^{P} \delta_j x_i^j \tag{A.5}$$

$$\delta_j = \sum_{o=1}^{I} sgn(C_o)(e^j_{n-1,o} - \bar{e}_{n-1,o})\frac{df_n(s^j_n)}{ds^j_n} \tag{A.6}$$

$$C_o = \sum_{j=1}^{P}(e^j_{n-1,o} - \bar{e}_{n-1,o})(f_n(s^j_n) - \bar{f}_n)$$

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

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

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

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

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.

Appendix B

A Quasi-Newton-based training algorithm

In the output-side training of the constructive OHL-FNNs and the polynomial OHL-FNNs (see Chapters 2 and 4 for details), computing the exact pseudo-inverse leads readily to an LS solution if the output nodes are considered to be linear. However, when constructive FNN is applied to classification problems, the output nodes of the networks have to be selected as nonlinear function. The output-side training now requires an algorithm that achieves a fast convergence and small SSE, at possibly the expense of increased computational load. In this thesis, a Quasi-Newton algorithm is utilized. A brief review of the scheme is given below.

The criterion for output-side training is given by

$$\begin{aligned} J_{output} &= \frac{1}{2}\sum_{o=1}^{I}\sum_{j=1}^{P}(y_{n,o}^{j} - d_o^j)^2 \\ y_{n,o}^{j} &= f_o(z_o^j), \\ z_o^j &= \sum_{i=0}^{n} v_{n,i}u_i^j \end{aligned} \tag{B.1}$$

where $y_{n,o}^{j}$ is the response of the o-th output, d_o^j is the target at the o-th output, u_i^j is the output of the i-th hidden unit, and j is the number of the training sample, $f_o(\cdot)$ is the activation function of the o-th output node. In the algorithm, the output-side weights $\mathbf{v}_n = (v_{n,0}, v_{n,1}, \cdots, v_{n,n})^T$ are updated according to

$$\mathbf{v}_n(t+1) = \mathbf{v}_n(t) - \mathbf{H}(t)\nabla J_{output}(\mathbf{v}_n(t)) \tag{B.2}$$

where $\nabla J_{output}(\mathbf{v}_n(t))$ is the current gradient vector given by

$$\nabla J_{output}(\mathbf{v}_n(t)) = \left[\frac{\partial J_{output}}{\partial v_{n,0}} \; \frac{\partial J_{output}}{\partial v_{n,1}} \; \cdots \; \frac{\partial J_{output}}{\partial v_{n,n}}\right]^T\Bigg|_{\mathbf{v}_n=\mathbf{v}_n(t)}, \tag{B.3}$$

$\mathbf{H}(t)$ is the approximate inverse Hessian calculated iteratively according to

$$\mathbf{H}(t+1) = \left(\mathbf{I} - \frac{\delta(t)\gamma(t)^T}{\delta(t)^T\gamma(t)}\right)\mathbf{H}(t)\left(\mathbf{I} - \frac{\delta(t)\gamma(t)^T}{\delta(t)^T\gamma(t)}\right)^T + \frac{\delta(t)\delta(t)^T}{\delta(t)^T\gamma(t)} \tag{B.4}$$

where

$$\delta(t) = \mathbf{v}_n(t+1) - \mathbf{v}_n(t), \tag{B.5}$$

$$\gamma(t) = \nabla J_{output}(\mathbf{v}_n(t+1)) - \nabla J_{output}(\mathbf{v}_n(t)), \tag{B.6}$$

and $\mathbf{I}$ is identity matrix of appropriate dimension.

Note that several types of Quasi-Newton algorithms are available in the literature (see [38] and references therein), and the algorithm presented above is one of them, which belongs to the class of second-order algorithms. Although, it is based on the Newton's method, however it does not require exact calculations of the second order derivatives (see (B.2)-(B.6)). Instead an approximate Hessian matrix in (B.4) is updated at each iteration of the algorithm. The update is computed as a function of the gradient (B.4)-(B.6).