STRUCTURAL ADAPTATION AND GENERALIZATION IN SUPERVISED FEED-FORWARD NETWORKS

Joydeep Ghosh and Kagan Tumer

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712-1084

This article appeared in:
Journal of Artificial Neural Networks, Vol 1, No. 4, pp 431-458, 1994.

Abstract

This work explores diverse techniques for improving the generalization ability of supervised feed-forward neural networks via structural adaptation, and introduces a new network structure with sparse connectivity. Pruning methods which start from a large network and proceed in trimming it until a satisfactory solution is reached, are studied first. Then, construction methods, which build a network from a simple initial configuration, are presented. A survey of related results from the disciplines of function approximation theory, nonparametric statistical inference and estimation theory leads to methods for principled architecture selection and estimation of prediction error. A network based on sparse connectivity is proposed as an alternative approach to adaptive networks. The generalization ability of this network is improved by partly decoupling the outputs. We perform numerical simulations and provide comparative results for both classification and regression problems to show the generalization abilities of the sparse network.

1 INTRODUCTION

An Artificial Neural Network (ANN) that does not have any feedback loops in the underlying signal flow graph, is referred to as a Feed-Forward Network (FFN). Popular FFNs that use supervised learning include multi-layered perceptrons (MLPs), and Radial Basis Function networks. Such supervised FFNs have been successfully applied to a wide variety of problems involving learning from examples, and are largely responsible for the recent popularity of ANN technology.

While the use of multilayered FFNs has been widespread, their inner workings are often not fully understood, resulting in "black box" type tools, with the network performance varying greatly depending on what is inside the box. Demystifying the box for any specific problem involves obtaining satisfactory solutions to questions of
(i) *appropriate network structure* (type; number and sizes of hidden layers; full/local/sparse connectivity),

(ii) *operational aspects* such as training modes, training data selection and choice of learning and momentum rates, and
(iii) *network evaluation* including interpretation of results, performance prediction on unknown samples with associated prediction confidence, and extraction of rules from a trained network.

Over the past few years, several methods have been proposed for obtaining a suitable network structure, either by an analysis of the data set or by incrementally growing/shrinking a given network. Also, ideas from disciplines such as function approximation and applied statistics have been applied to guide network structure selection and network evaluation. This article summarizes some of these developments using a common framework centered around the notion of generalization in supervised neural networks, and introduces the sparsely connected network as an alternative to structurally adaptive networks. We also present experimental results that show good generalization ability for simple sparse networks.

1.1 Generalization

One of the most important features of learning systems is their ability to "predict" the outcome of a certain event based on "experience." This experience is provided by the (possibly repeated) presentation of a *training set* of input-output mappings. The performance of the network is then evaluated on a *test set*, a data set that is disjoint from the training set. The trained network is expected to predict the correct input-output relationship for the data in the test set by *generalizing* the learned mapping. Clearly there is an underlying assumption that the training set is representative of the test set, for example, that they are both extracted from the same unknown distribution, or are an outcome of the same physical phenomena.

The prevalent definition for generalization ability then, is the performance of the network on the test set, or more strictly, the *expected* performance on test sets for the specified problem. A learning system that can generalize well is extremely useful as it is flexible and robust when faced with new and/or noisy data [1].

In general, the number of possible generalizations from a small training data set is immense. This poses a serious problem in analyzing and interpreting the performance of an FFN. "Common sense" as we perceive it, is generalization within certain constraints that is some sense reflect our expectations. This does not make the other generalizations "wrong," it simply emphasizes that there are many possibilities, and that "correctness" depends on the framework one chooses to adopt.

1.2 Efficient Feed-Forward Network Designs

The generalization ability of an FFN is influenced by its size. For example, an MLP network with too few hidden units (under-determined) does not learn the required tasks, while one with too many hidden units (over-determined)

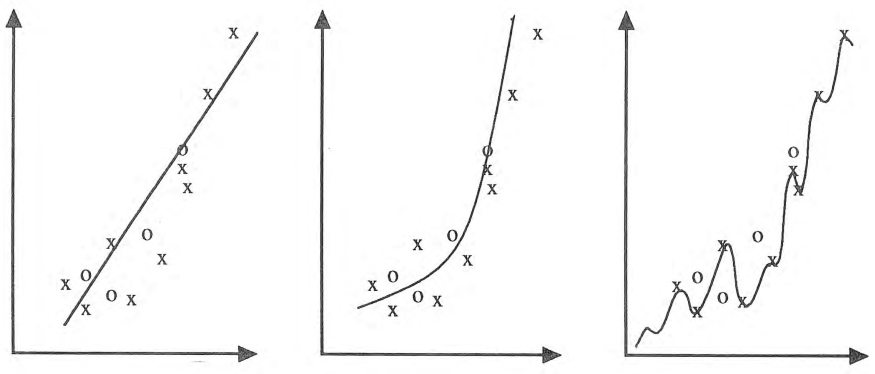

Figure 1: (a) Under-fit data: Poor fit on training set. (b) A good fit. (c) Over-fit data: Perfect fit on training set, but poor fit on test patterns. (x's represent training patterns, o's represent testing patterns.)

usually generalizes poorly[2]. Such behavior is qualitatively similar to curve fitting, where the right complexity is needed for a good fit to both training and test data, as illustrated in Figure 1. This observation motivates the following definition:
An FFN design methodology is called *efficient* for a given mapping problem, if it results in a network architecture whose structural and functional characteristics lead to quick training and good generalization for that problem.

This paper presents and evaluates diverse approaches, theoretical and heuristic, for obtaining *efficient* FFN design methodologies. First, we note that FFNs are applied to two broad classes of problems:

- **Classification:** discrete-discrete and continuous-discrete mappings.
- **Regression:** discrete-continuous and continuous-continuous mappings.

Classification consists of a decision making task where the output fits into well defined categories. The performance of the network in this case can be easily evaluated. The output is compared to the target, and is either correct or incorrect. Regression problems on the other hand, involve approximating a continuous-valued target function. In this type of problem, the performance of the network has to be gauged by a suitable distance metric such as mean square error (MSE) [3].

Section 2 discusses different possibilities with static MLP networks and introduces the sparsely connected network, which is designed to perform in noisy and high dimensional environments. Section 3 studies dynamic approaches that have been used to obtain efficient networks. "Pruning" methods, which progressively remove weights and/or connections until a desirable performance level is

reached, are discussed first. Then, "construction" algorithms, which start with a limited number of units and add layers and/or units to achieve the desired performance level, are presented. Section 4 examines several theoretical frameworks dealing with the generalization issue and with the estimation of prediction error. Section 5 provides a comparison of certain algorithms discussed in the previous sections. These results are based on simulations performed on both regression and classification problems.

2 STATIC ARCHITECTURES

2.1 Generic MLP Networks

Figure 2 depicts the generic structure of an MLP network with a single hidden layer. The responses of the k^{th} hidden unit, h_k, and the j_{th} output, o_j are given by

$$h_k = g(\sum_i v_{ki} x_i); \quad o_j = f(\sum_k w_{jk} h_k), \tag{1}$$

where the indices i and k sum over the input and hidden units respectively. If the activation function $g(\cdot)$ is sigmoidal and $f(\cdot)$ is linear, then the network can uniformly approximate any continuous function provided sufficiently many hidden units are used [4]. Moreover, the L_2 norm of the approximation error by an MLP with h hidden units is bounded by $2C_f/h^{1/2}$, where C_f denotes the first moment of the Fourier magnitude distribution of the function to be approximated [5]. Interestingly, this bound is independent of input dimension d. These two properties partly explain why MLPs have been successfully applied to a wide range of function approximation problems and have often eluded the "curse of dimensionality."

Learning in the network of Figure 2 entails adapting the weights (v_{ki}s and w_{jk}s) in order to minimize a cost function such as the Mean Square Error (MSE) function:

$$E \;=\; \frac{1}{2}\sum_{j,p} ({t_j}^p - {o_j}^p)^2 \tag{2}$$

where ${t_j}^p$ is the target value for the p^{th} pattern, and ${o_j}^p$ is the actual output for the p^{th} pattern.

The well known back–propagation algorithm for training MLPs is a generalization of the *delta rule*, and is obtained by performing steepest descent on the "instantaneous gradient" of E in weight space [6, 7]. The use of differentiable activation functions for the hidden units allows the use of the chain rule for updating the weights of links that are not directly connected to an output unit. Each weight can then be updated using

$$\Delta w_{jk} = \eta \, \delta_j \, o_k \tag{3}$$

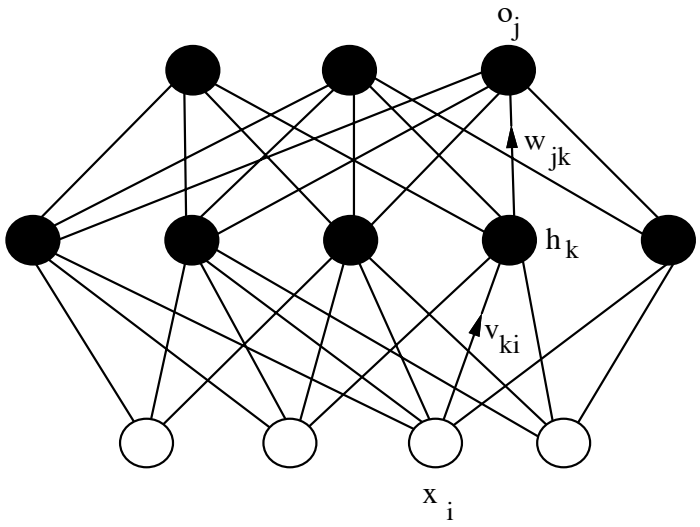

Figure 2: A Feed-forward Network with one hidden layer.

where η is the learning rate, and $\delta_j = (t_j - o_j)f'(h_j)$ for the output layer, and $\delta_k = \sum_j w_{jk}\delta_j$ for all other layers with the k^{th} layer preceding the j^{th} layer. Details of the "back–propagation of error" learning rule, and of various methods for more effective gradient descent can be found in many textbooks [8, 3]. We note that even with the use of sophisticated techniques of conjugate gradient and second-order methods, many iterations may be required for training. The main problem is in the structure of the algorithm, a gradual search of the weight space with susceptibility to local minima, rather than in the specific way of implementation. The default MLP network introduced earlier in this section is "fully connected" in the sense that all the units in one layer are connected to all the units in the subsequent layer. However there are situations where more restricted connectivity could be sufficient or even preferable. For example, Rosenblatt studied "diameter limited" perceptrons in which each output cell received input only from a localized group of input cells forming a visual receptive field [9]. A follow-up on the biologically motivated idea of local connections is the idea of weight replication. In this scheme, each unit of a layer is connected to a limited number of units from the preceding layer by the same weights. By replicating or sharing weights, the number of free parameters in a network is reduced, which may increase the training speed and improve generalization abilities [10]. Weight duplication also introduces bias in the network and can be exploited to enforce some invariance (shift, rotation) in the network response. The theoretical relationship between biased and unbiased networks is discussed in more detail in Section 4.2.

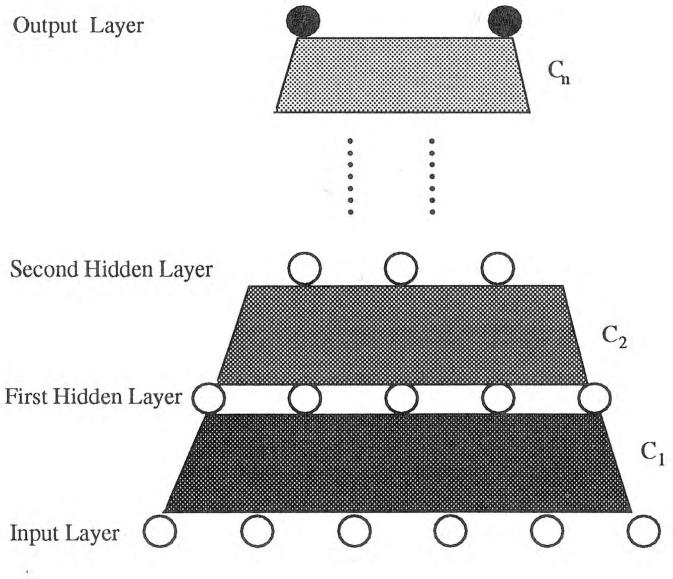

Figure 3: Sparsely connected network with $C_1 \geq C_2 \geq \cdots \geq C_n$, where C_i's refer to connectivity between layers.

2.2 Sparse Connectivity

In this section we introduce the sparsely connected network as an alternative to a locally connected network. In this scheme, not all units from a layer are connected to all units in the subsequent layer, yet the connections are not localized. The restrictions of a diameter limited perceptron do not apply to this type of network.

Suppose we were to remove certain connections *before* training starts. The advantages of the resulting sparsely connected network are twofold. Due to the "freedom" gained by the hidden units, the outputs are more decoupled. Moreover, the number of parameters is reduced. From theoretical considerations elaborated on in Section 4 (also see [2, 11]), we expect these two features to lead to better generalization and reduced sensitivity to noise.

The connections between the input layer and the first hidden layer is where the feature extraction takes place. Therefore it is advisable to keep the connectivity of this layer close to full. The subsequent layers can then have decreasing connectivity until the outputs are clearly separated. Let the probability that two units in layers i and j are connected be denoted by C_i. Figure 3 shows the structure of a sparsely connected network built with these specifications. The shading of the weights indicate the degree of connectivity with lighter shades having lower connectivity than the darker ones. Such a network will be experi-

mentally evaluated in Section 5, and compared with schemes that start with a fully connected network and remove weights during or after training.

3 STRUCTURAL ADAPTATION

The fully-connected, locally connected and weight replicated network structures described in the previous section are called static because the number of units and connections are predetermined. The learning algorithm consists of finding a set of weights that minimizes the error function. In this section, we survey networks with dynamic structures where the learning algorithms not only search the weights space, but also modify the architecture of the network during training.

3.1 Pruning Methods

Pruning methods start from initial architecture that is overparameterized, and selectively remove units and/or connections during training, until a satisfactory performance level is reached. The pruning phases may be interleaved with retraining of the pruned network. The following heuristic methods attempt to reach an efficient network topology using this approach (also see [12]).

3.1.1 Weight Removal and Optimal Brain Damage

A first cut at pruning a trained network is simply to delete weights with small magnitude, as they are expected to have the least influence. However, there are several problems with this method. First, the assumption that a weight's importance is directly proportional to its magnitude is a gross approximation. For example, a hidden unit whose net input activation is very low or very high (i.e. which is in a saturation region) will be less sensitive to an afferent weight as opposed to another unit whose net input activation is near zero. Second, the threshold for deletion overly influences the outcome and has to be fine-tuned, rendering the method difficult to apply.

The actual effect of the deletion of a weight can be quantified if the cost function is recomputed after that weight has been removed. This method though, is impractical due to the high cost associated with computing the MSE at each step. A method called Optimal Brain Damage (OBD) [13] attempts to circumvent this problem by analytically determining the relative importance of weights in an FFN, using second derivative information. OBD operates by building a local model for the cost function using a Taylor series. The effect of a perturbation δw_i of weight w_i results in a change in the objective function E, by

$$\delta E \approx \sum_i \frac{\partial E}{\partial w_i}\,\delta w_i + \frac{1}{2}\sum_i h_{ii}\,\delta w_i^2 \tag{4}$$

where $h_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$. In Equation 4, cross terms in the Hessian (h_{ij}'s), and terms of order higher than two are neglected. Furthermore, if the perturbation

is performed after the network has reached a local minima, the first term of Equation 4 can be ignored. This motivates defining saliency s_i of a weight w_i as $s_i = h_{ii} \frac{w_i^2}{2}$. Geometrically, h_{ii} denotes the convexity of the error surface in weight space about the local minima, along the i^{th} weight. Weights with small saliencies are those which influence the objective function the least and are prime candidates for removal. Thus the OBD method involves training a network that is sufficiently large, and then removing those weights whose saliencies are below a threshold. The pruned network can be further trained and pruned if necessary. The speed of this algorithm can be increased if the perturbations are performed on sets of weights instead of individual weights. A more general use of the Hessian matrix leads to the Optimal Brain Surgeon (OBS) algorithm, where the h_{ij} terms are not assumed to be zero [14].

3.1.2 Weight Decay

Weight decay is another method that concentrates on reducing the number of parameters in a network. The essential idea behind this approach is the gradual removal of connections that are not frequently reinforced during training [15]. One way to implement this scheme is to allow each weight to decay towards zero by adding a negative term to Equation 3, that is proportional to the current weight. The new update equation becomes

$$\Delta w_{jk} = \eta(\delta_j o_k - \lambda w_{jk}) \,, \tag{5}$$

where λ is the decay parameter. This updating scheme incorporates weight decay by reducing weights by $\Delta w = -\eta\lambda w$, unless they are reinforced.

The modified updating rule of Equation 5 can be also obtained through gradient descent on a biased cost function $E' = E + \lambda P$, obtained by adding to the cost function E, a *penalty term* $P = \sum_{j,k} w_{jk}^2$, that is weighted by λ. We shall see later that this can be viewed as a method of complexity regularization.

The main drawback of this simple method is that it overly penalizes large weights. A more suitable approach is to select

$$P = \sum_{j,k} \frac{w_{jk}^2}{w_0^2 + w_{jk}^2}.$$

This term contributes $\frac{-2\lambda w_{jk}\, w_0^2}{(w_0^2+w_{jk}^2)^2}$ to the weight update, and penalizes weights around w_0.

The advantage of weight decay is that the training time is not increased by requiring the network to be retrained after weight removals. Learning and pruning take place simultaneously, since the updating itself incorporates the decay mechanism. Numerical simulation results report improved generalization ability of networks trained with weight decay [16].

A variation on weight decay is weight group decay, which can lead to the removal of entire hidden units [17]. Here a bias term similar to that in Equation 6 is defined as either $\frac{w_k}{1+\lambda w_k}$, or $(1 - e^{-\lambda w_k})$ where $w_k = \sum_j |w_{jk}|$. This method

isolates weight updates to groups of weights that can be traced to specific hidden units.

3.1.3 Suppression of Hidden Unit Activations

Another approach to reducing the size of a network is to suppress hidden units rather than weights, by choosing the penalty term P of Section 3.1.2 to be of the form $\sum_{k,l} e({o_{kl}}^2)$ where $e(\cdot)$ is a positive monotonic function, and o_{kl} is the output of the k^{th} hidden unit in the l^{th} hidden layer. Gradient descent is then performed on the modified cost function.

When a unit's activation summed over the entire training set falls below a certain limit, that unit is removed. The essential idea behind this method is that some of the units in an FFN may be redundant. By placing constraints on the weight space, this method forces the hidden units to efficiently represent the input features, and therefore results in a reduction in the number of hidden units needed.

This method is sensitive to the selection of the function $e(x)$, which can simply be x or a nonlinear functions, such as $e(x) = \frac{x}{1+x}$ or $e(x) = \log(1+x)$. When applied to the "exclusive or" problem (XOR), Chauvin reports [18] that the activation of all but two hidden units were suppressed, essentially removing the others. This method was also tested on a phonetic labeling task, and was reported to reduce the number of hidden units from twelve to three, without significantly affecting the performance.

3.2 Construction Methods

An opposing approach concentrates on incrementally building networks starting from simple structures. The resulting architectures are more varied than the ones obtained by pruning large networks, since one has more options while adding new units.

Dynamic Node Creation, introduced by Ash [19], is designed to add hidden units to the network whenever the learning appears to slow down. A small network is first trained using the back–propagation algorithm. When the slope of the MSE vs. number of epochs curve falls below a certain threshold, a new node is added. The network is then retrained and the process continues until the MSE reaches a satisfactory value or starts curving upwards.

The use of this method is limited to a network with one hidden layer, since there are no contingencies for creating new layers. The network has to relearn after each node creation, increasing the training time considerably. Furthermore, the final network is prone to overfitting the training data.

Cascade-correlation, introduced by Fahlman and Lebiere [20], builds a multi-layered FFN starting from a one-layer network. The most important feature of this approach is an incremental construction scheme that avoids the back–propagation of the error through several layers, and thus reduces the training time. The starting point of this algorithm is a one-layer FFN, which is

trained with the simple delta rule. When the performance cannot be significantly improved upon, a new unit is added that receives connections from the input units and all the currently present hidden units, as shown in Figure 4. The connections to this hidden unit are calculated *before* it is connected to the network, in a way which maximizes its usefulness. Details can be found in [20, 8, 21].

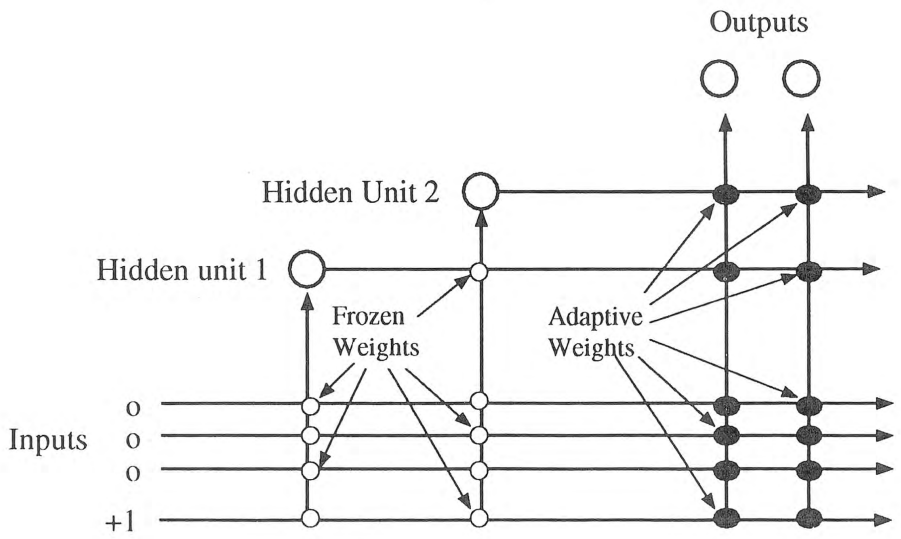

Figure 4: Cascade-correlation network [20].

Dynamic node creation and cascade-correlation are special cases of data-dependent incremental model construction techniques that are studied in approximation theory. For example, constructive polynomial-based function approximation is provided by the GMDH algorithm [22], the self-organizing neural network [23], the ridge polynomial network [24] and the LMS tree of Sanger [25], among others. Even in the context of MLP type networks, other heuristics such as splitting nodes based on principal component analysis of oscillating weight vectors [26], can be found in recent literature.

Another class of constructive algorithms depend on the performance of certain units at intermediate stages, and are only applicable to classification type problems. The tiling algorithm and the upstart algorithm for example build a network by incrementally adding units and layers [3].

3.3 Incremental Construction of Radial Basis Function Networks

Multi-layered perceptrons are not the only type of supervised FFNs for which structural adaptation algorithms exist. RBF networks, a class of single hidden-layer feed-forward networks in which radially symmetric basis functions are used as the activation functions for the hidden layer units, are becoming popular particularly for localized or low-dimensional problems. RBF networks are primarily aimed at multivariate function interpolation or function approximation, and have been used successfully for problems such as prediction of chaotic time series[27]. They also serve as universal approximators using only a single hidden layer [28, 29].

For RBF networks, a constructive algorithm involves incrementally adding kernel nodes, and possibly adjusting the widths along with the weights. A particularly effective method is imbibed in the resource-allocation network (RAN) of Platt [30]. RAN works by either performing gradient descent or creating a new kernel, depending on the current training pattern. If the network performs well on the pattern, the weights are adjusted using standard LMS descent. If, on the other hand, the performance is deemed poor, a new unit is added, and the pattern is "memorized."

RAN is tailored to on-line training. When the entire training data set is available, an effective method for incremental growth is provided by the hierarchically self-organizing learning technique of Lee and Kil [31]. The algorithm incrementally (and stingily) recruits centroids based on control of effective radii of influence (boundaries) of individual centroids with Gaussian receptive fields. It also adapts the locations and widths of the centroids using gradient descent. Each centroid has an accommodation boundary and a class representation. If a training sample not within the accommodation boundary of some centroid with the same class label, a new centroid is created. Also, if MSE curve flattens out, the boundaries of all units are shortened. Otherwise, the parameters are adapted using gradient descent. A notable feature of this approach is that it changes both dimension and shape of error surface during training, thus making it easier to escape local minima.

4 THEORETICAL CONSIDERATIONS

This section deals with different theoretical approaches for determining the requirements on the architecture of FFNs, and the selection of training sets for achieving good generalization. These methods are obtained from the related areas of nonparametric statistical inference, statistical pattern recognition, estimation theory and the classical theory of functions. They provide useful insights into the behavior of FFNs, and provide more systematic alternatives to some of the heuristics presented in the previous section. The first three disciplines also provide several well-studied techniques for selection of training data and for estimating performance on test data in classification problems. These include

density estimation, cross-validation, resampling and bootstrap methods. Since such techniques are treated in several books [32], they will not be discussed here.

To begin, observe that FFNs as a general class are part of nonparametric regression models [2]. Of course, if a particular network is chosen and its architecture is fixed, the network becomes a specific parametric scheme, since only the weights (the parameters) have to be determined.

First, an account of learning as function approximation is presented to evoke parallels between the two concepts. Second, the statistical properties of learning are discussed in order to formalize the problems that are involved in FFN designs, including the fundamental bias-variance tradeoff. This leads to the concept of complexity regularization, which can be directly applied to obtain suitable network structures. Finally, the complexity of functions realizable by a network class lead to quantifiable results on their generalization ability, and on the prediction of the test set error rate. Some of these issues are also examined in [33, 8, 34].

4.1 Learning as Function Approximation

The problem of function approximation consists of representing an unknown function $f(X)$ by $F(W, X)$, an approximating function that uses a set of parameters W. The task consists of either finding a set W for a given function F, or of finding both F and W in order to perform the best possible approximation. Classical approximation theory provides well established results for this type of query [35, 36]. Observe that the process of searching for the "right" functional form and/or parameters is equivalent to learning if the function is realized through a neural network based on samples of the input-output map [37].

Function approximation can be expressed formally as the search for the set of parameters W' which satisfies

$$d[F(W', X), f(X)] \;\leq\; d[F(W, X), f(X)] \tag{6}$$

for all W, where d is a suitable distance metric, and X is the multi-dimensional input. The solution W' satisfying Equation 6 is referred to as the *best approximation*. The likelihood of finding the best approximation depends on both the class of functions to which $F(W, X)$ belongs, and the original function $f(X)$.

The analogy with learning in FFNs is clear on considering the following two forms of F for approximation:

- The linear approximation in a suitable basis $\{Z_i\}$, which characterizes a two-layer FFN with linear output units, such as the RBF network:

$$F(W, X) \;=\; \sum_i W_i Z_i(X) \tag{7}$$

- The nested sigmoids method, which corresponds to a multi-layered FFN,

such as the MLP:

$$F(W, X) \;=\; g(\sum_j \; v_j \, g(\cdots \; g(\sum_i \; w_i \, x_i) \; \cdots)) \tag{8}$$

where g is a sigmoidal function, and the weights are $W = \{\cdots, v_i, \cdots, w_i, \cdots\}$.

Note that the type of nested nonlinear functions expressed by Equation 8 is unusual in classical approximation theory, and the results on the powerful approximation capabilities of such forms, for a single hidden layer of sigmoidal units, are very recent [33, 38, 4, 37].

Pictorially, function approximation can be viewed as reconstructing a surface (desired function) from sparse sample points (training set). Pursuing this analogy, generalization can be defined as estimating the height of the surface (output value) at a point in space where no examples are available. If the surface is reconstructed faithfully, generalization ability will be good, and the system will have "learned" the mapping. Clearly, generalization is based on the assumption that there is redundant information in the data. If a mapping is totally random, generalization is not possible since there is no regularity that can be learned.

4.2 Statistical Framework

Given the approximation theory framework presented above, the immediate question that arises is, by choosing F and W appropriately, how well can a network perform based on a given training set. Are there any fundamental limits to performance against test data? Note that the training and test sets are considered as random samples from the same but unknown distribution, and that the observations are typically noisy. So it is not surprising that some answers to the questions above are provided by statistical estimation theory [2].

As mentioned earlier, the main objective in supervised learning is to accurately estimate a function $f(X)$ from repeated observations of (X, y) pairs, where $y \;=\; f(X) +$ noise, if any. For a minimum squared error criterion, the regression of y on X, $E[y|X] \equiv F^*(X)$, is the best predictor of y given X, since for any function F

$$E[(y - F(X))^2|X] \;\;\geq\;\; E[(y - E[y|X])^2|X] \tag{9}$$

(for proof see Geman *et al* [2]).

Thus, a valid goal is to minimize an error function

$$r(F, F^*) = E[(F(X) - E[y|X])^2]. \tag{10}$$

This error can be further divided into two parts, called "bias" and "variance," respectively [2]:

$$E[(F(X) - E[y|X])^2] \;=\; (E[F(X)] - E[y|X])^2 \;+\; E[(F(X) - E[F(X)])^2] \; . \tag{11}$$

Reducing the total error is a tricky process, because a model-free estimator with low bias leads to high variance, and an incorrect model leads to high bias. This is similar to the problem encountered while predicting the size of an FFN. Too few hidden units result in poor learning because it corresponds to an inadequate model with high bias. Too many hidden units, on the other hand, prevent generalization, because the excess of parameters leads to high variance. Equation 11 then demonstrates the heart of the problem: *How to reduce bias without increasing variance or how to reduce variance without increasing bias.*

In order to avoid this dilemma the network has to be large enough not to introduce an initial bias, and the training set has to be large enough to overcome any variance introduced by individual sample points. The goal is to reduce the variance progressively without introducing too much bias, and is the rationale behind the pruning methods of Section 3.1, and regularization, addressed in the next section. Alas, the training set is often limited in size in the real world. The biases and added uncertainties due to small sample size have been studied in [39].

An important issue in statistics is that of asymptotic convergence, or "consistency." Most non-parametric regression algorithms—including FFNs—are consistent, i.e. they would approximate the objective function, to an arbitrary degree of accuracy given unlimited data. The question then becomes, is there a large enough training set that allows learning in practice? The answer given by Geman *et al* [2] is that for truly model-free estimators, learning is a practical impossibility. Yet FFNs in general are never truly model-free. The architecture, the size, the connection scheme, and even the selection of the data set often conveys considerable information, and introduces bias. Satisfactory performance is then attainable with a finite set of training patterns, since some a priori knowledge is encoded before purely statistical learning begins.

4.3 Regularization

The statistical framework provided a theoretical basis for separating the error due to the model (bias) and the error due to the data (variance). If a model-free estimator is used to minimize MSE over the training set, the resulting model will be too complex and lead to high variance and thus high total error, leading to poorer performance on the test set. In other words, increasing the complexity of the model improves the training set error at the expense of the generalization ability.

Complexity regularization counters this problem by choosing the network estimate to minimize a criteria of the form $\sum_p (y^p - F(X^p))^2 + \lambda P(F)$ where λ is positive and determines the amount of regularization, and $P(F)$ is a non-negative penalty term that uses a priori knowledge about the mapping to constrain the solution. For example $P(F) = \| \int_R \nabla^2 F \|^2$ penalizes approximating functions F that are not smooth. The penalty functions corresponding to the weight decay schemes of Section 3.1, on the other hand, prefer networks with smaller or lesser weights. They embody the "Occam's razor principle" which states that unnecessarily complicated models should not be selected over

simpler models.

A more rigorous approach shows that for squared error and log-likelihood loss functions and given n training samples, complexity regularization indicates selecting an estimating function [40]

$$\widehat{F} = argmin_F(\frac{1}{n}\sum_{p=1}^{n} d(y^p, F(X^p)) + \lambda\frac{1}{n}C_n(F)), \tag{12}$$

where d is the distance function and F is searched from the class of functions considered. The measure of network complexity, C_n, is essentially the logarithm of the number of functions required to approximate functions in the class to within a prescribed accuracy. Moreover, the estimation error is bounded in probability by $r(F, F^*) + C_n(F)/n$ (for squared error or log-likelihood loss functions), where r() is given by Equation 10. For more details, see [5, 40]. The results indicate that the one can afford more complex estimators if the training set is larger.

Regularization theory has been used to form *regularization networks*, FFNs with a single hidden layer [37]. These networks use a variety of activation functions for the hidden layer, one of which can be a Gaussian, making RBFs with Gaussian kernels a subset of regularization networks. Complexity regularization is difficult to implement directly, as both λ and $C_n(f)$ have to determined. The Bayesian framework alleviates this problem by tuning the parameters (including λ) *during* training [41, 42].

4.4 Expected Test Set Error and Risk Minimization

We now focus our attention to the test set error, and establish the "prediction risk," a measure of the generalization ability of the network. In many practical situations, only the training set error is known, but of course the most important measure is the performance on test set which is presumably sampled from the same distribution as the training set, but is otherwise unknown. If a bound can be imposed on the difference between the test set and training set errors, the level of trust one can place on network results can be known. Also, since varying the test set can lead to different generalization performance measures, and it becomes apparent that a method independent of the test set is needed to determine the expected performance on it.

Equation 10 gives an indication that the difference between training and test error is related to model complexity. In a linear model the test set error ε_{test} is related to the training set error ε_{train} by

$$\langle \varepsilon_{test} \rangle \;=\; \langle \varepsilon_{train} \rangle + 2\sigma^2 \frac{p}{n} \tag{13}$$

where σ^2 is the noise variance, p is the number of parameters, n is the number of training patterns, and $\langle \cdot \rangle$ is the expected value operator [35]. The basic tradeoff between the number of parameters in the network and the training set error is illustrated in Equation 13. Increasing the number of free parameters

in the network initially decreases $\langle \varepsilon_{test} \rangle$ as the reduction in $\langle \varepsilon_{train} \rangle$ counteracts the increase in the second term of Equation 13. Further increase in the number of parameters provides only a modest decrease in $\langle \varepsilon_{train} \rangle$ if any, resulting in an increase in the overall test set error.

For a nonlinear system such as MLP, Moody has shown that a similar relation can be obtained with a second order approximation [11]:

$$\langle \varepsilon_{test}(\lambda) \rangle \approx \langle \varepsilon_{train}(\lambda) \rangle + 2\sigma_{eff}^2 \frac{p_{eff}(\lambda)}{n} \tag{14}$$

where p_{eff} and σ_{eff}^2 are the "effective" number of parameters and the effective noise variance respectively, and λ is the regularization parameter as in Equations 12, emphasizing the importance of proper tuning. Moody also shows that for an MLP, the effective number of parameters increases sublinearly with the number of weights, thus providing more insight into the robustness of even fairly large MLP networks. Moreover, Equation 14 provides a Generalized Prediction Error measure that can be minimized while training the network, thus extending prediction error criteria developed for linear estimators. Obtaining the values of the effective parameters and a suitable choice of λ for a given problem however, is not trivial.

4.5 Results based on VC Dimensionality

Since network architectures are directly related to the types of functions they can successfully implement, it would be beneficial if the complexity of function classes could be quantified. Both Equation 10 and 14 indicate that a "complexity parameter" could be used to predict the difference between training set and test set errors. The Vapnik-Chervonenkis (VC) dimension, d_{vc}, is such a parameter and provides bounds on the generalization ability of network families in terms of the classes of functions they can realize [43, 44]. Due to the difficulty involved with estimating the VC dimension, its use is generally restricted to classification problems, even though the theory encompasses both binary and continuous functions.

It can be shown that [43]:

$$\int E(y, F(X, W))\, dP(X, y) < \frac{1}{n} \sum_{i=1}^{n} E(y_i, F(X_i, W)) + C_0(\frac{n}{d_{vc}}, \eta) \tag{15}$$

with probability $1 - \eta$, where $C_0(\frac{n}{d_{vc}}, \eta)$ is a confidence interval depending on the training set size and the VC dimension of the class of functions under investigation. In the above equation, E is an error measure, y is the actual output, $F(X, W)$ is the approximating function, and $P(X, y)$ is the joint probability distribution. Thus the left hand side is the expected value of the error, and the first term on the right, called the empirical risk, is the error measured over the training set. The VC dimension used in this manner provides a conservative bound for the generalization error.

5 EXPERIMENTAL RESULTS

In this section, we report the results of the numerical simulations that we conducted to examine the learning and generalization properties of different networks. The experiments were based on two regression problems and one classification problem. The chosen problems had enough regularities to be "learnable," but were complex enough to provide a good measure of the abilities of different algorithms.

5.1 Comparative Studies with Regression Problems

5.1.1 The Gabor Function

The first regression problem consists of approximating a 2-D Gabor function. It is well known that the highly-oriented simple cell receptive fields in the visual cortex of mammals can be closely modeled by 2-D Gabor functions, which are Gaussians modulated by sinusoidal functions. Gabor functions play an important role in physics, since they are the unique functions that achieve the lower bound for the space-frequency uncertainty product, which is a measure of a function's simultaneous localization in both spatial and frequency domains [45]. The convolution version of *complex* 2-D Gabor functions have the following form:

$$G(x,y) = \frac{1}{2\pi\lambda\sigma^2} \cdot e^{-\frac{(x/\lambda)^2+y^2}{2\sigma^2}} \cdot e^{2\pi i(u_0 x + v_0 y)}. \tag{16}$$

Here, λ is the aspect ratio, σ is the scale factor and (u_0, v_0) are modulation parameters.

For simulation, the cosine function was used for the modulation and the parameters were set as follows; $\lambda = 1, \sigma = 0.5, u_0 = 1$ and $v_0 = 1$. Thus the actual function to be approximated was:

$$G(x,y) = \frac{1}{2\pi(0.5)^2} \cdot e^{-\frac{x^2+y^2}{2(0.5)^2}} \cdot \cos(2\pi(x+y)). \tag{17}$$

Then, 256 input points were selected from an evenly spaced 16×16 grid on $-0.5 \leq x \leq 0.5$ and $-0.5 \leq y \leq 0.5$. For training, 64 input points were randomly selected from these 256 points. The remaining 192 points were used for testing. Since Equation 17 has both positive and negative values, the hyperbolic tangent was used as the activation function.

The first network that was used was a fully connected MLP with two inputs, one output, and a single hidden layer consisting of $10, 20, 40$, or 80 hidden units. The training scheme consisted of the back–propagation algorithm with a momentum term. Figure 5 shows the results obtained with a noiseless data set, and Figure 6 shows the results for a data set with 20% noise (additive). In the absence of noise the networks with 20 and 40 hidden units performed better than larger or smaller networks, but the differences were quite small. In the presence of noise, the small network (10 hidden units) performed poorly. The

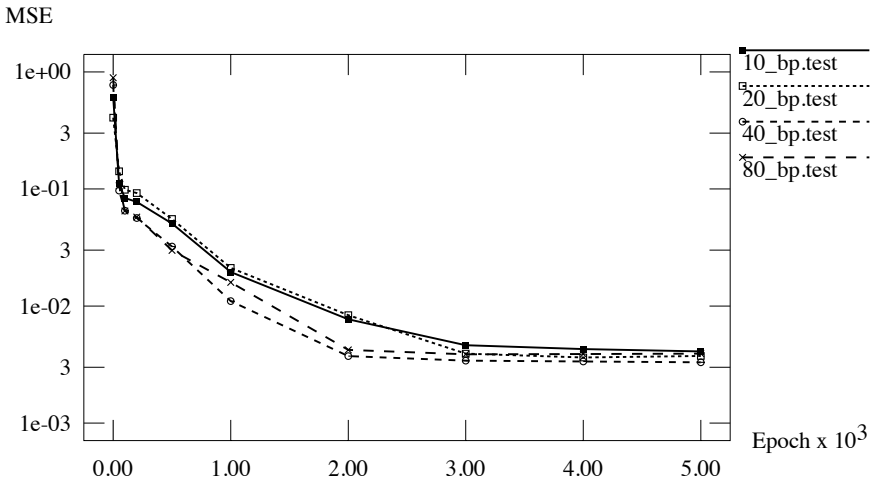

Figure 5: Back–propagation algorithm for the Gabor function with noiseless data: Effect of network size.

network with 20 hidden units proved to be the best as larger networks tended to overfit the data on the training set and had difficulty with generalization.

The next network was trained using Optimal Brain Damage (OBD). The major difficulty in applying this algorithm arose while selecting the thresholds for weight removal. The results were very sensitive to minor adjustments in this parameter and made the process difficult to automate. After changing the threshold several times, it became clear that with large networks, the algorithm would be difficult to fine-tune. The results presented were obtained with approximately 25% of the weights removed. Figure 7 compares OBD with back–propagation for a network with 80 hidden units. The OBD algorithm fared poorly for smaller networks.

The algorithm based on the suppression of hidden units activation presented in Section 3.1.3 was also implemented. It performed as expected for smaller problems, where it suppressed the activation of all but a few hidden units, but had trouble with the larger problems. As the number of hidden units and the number of training patterns increased, the algorithm's effectiveness decreased. The algorithm performs gradient descent on the following error surface:

$$E = \mu_{error}\, Error \; + \; \lambda\, P \tag{18}$$

where the terms $Error$ and $Penalty$, P, are discussed in Sections 2.1 and 3.1.3, respectively. Selecting λ too large prevented the descent on the error surface by suppressing all hidden units, and selecting it too small had no appreciable effect on the hidden units' activation.

The next algorithm tested was the sparsely connected network which has a static architecture, and was presented in Section 2.2. Figures 8 and 9 show the

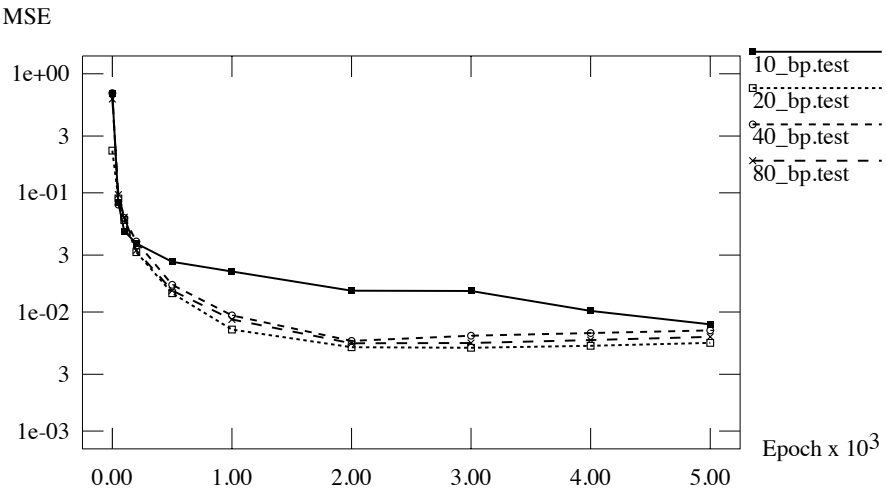

Figure 6: Back–propagation algorithm for the Gabor function with noisy data: Effect of network size.

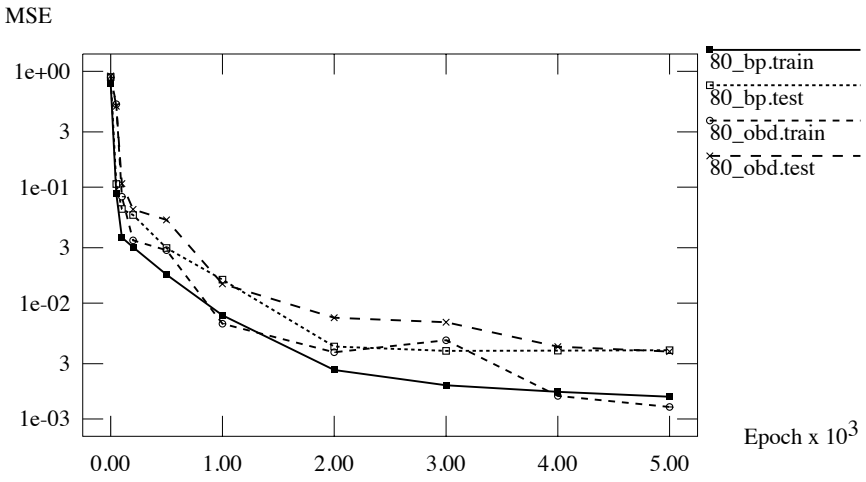

Figure 7: Back–propagation vs. OBD. (80 Hidden units.)

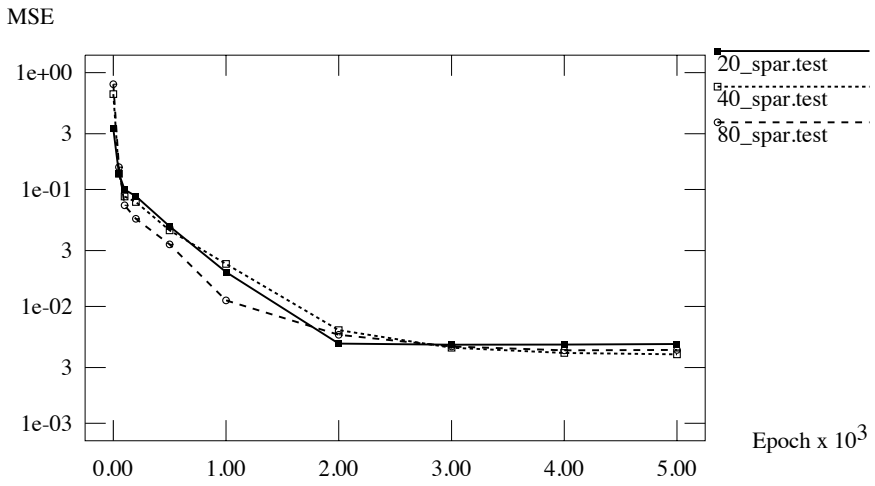

Figure 8: Sparse connectivity, with noiseless data.

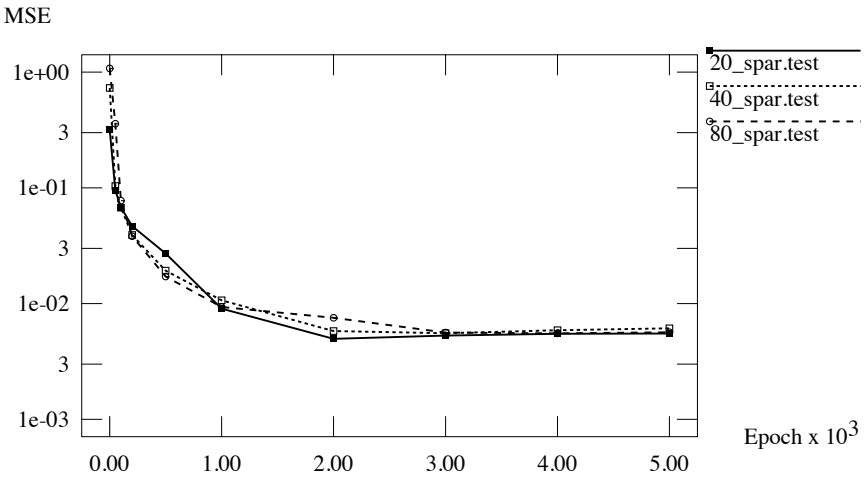

Figure 9: Sparse connectivity, with noisy data.

performance of the sparsely connected network. In the presence of noise the choice of the size of the network is not as critical as for other learning schemes. Moreover the generalization ability of the sparsely connected network is better than the generalization ability of the simple back–propagation network in the presence of noise.

Table 1: Comparative Results for the Gabor Function.

Training Algorithm	Noiseless data		Noisy Data	
	Training Set MSE	Test Set MSE	Training Set MSE	Test Set MSE
Back-Propagation				
(20 Hidden Units)	0.00152	0.00373	0.00322	0.00501
(40 Hidden Units)	0.00130	0.00334	0.00296	0.00571
(80 Hidden Units)	0.00151	0.00342	0.00288	0.00583
OBD				
(20 Hidden Units)	0.00182	0.00472	0.00683	0.00683
(40 Hidden Units)	0.00142	0.00316	0.00452	0.00630
(80 Hidden Units)	0.00129	0.00392	0.00543	0.00603
Sparse Connectivity				
(20 Hidden Units)	0.00137	0.00472	0.00432	0.00496
(40 Hidden Units)	0.00132	0.00382	0.00401	0.00554
(80 Hidden Units)	0.00161	0.00421	0.00397	0.00553

Table 1 shows how each algorithm performed on both the noiseless and the noisy data sets[1]. The results indicate that in the absence of noise, the back–propagation algorithm applied to a fully connected MLP is the best alternative. Even though OBD provided slightly better generalization for certain network sizes, the improvements were not significant enough to warrant the increased complexity. When noise is added, the sparsely connected network gives better generalization than both back–propagation and OBD algorithms.

5.1.2 Electrical Log Inversion Problem

The second regression problem is that of estimating formation resistivities through an instrument placed in a borehole. When the measuring instrument is in the borehole, the electrodes placed on its surface record different potentials which characterize the formation. Computing these potentials given the formation resistivities is the forward modeling part and is well formulated. Computing the formation resistivities from a set of electrode potentials, on the other hand, is the inverse modeling problem which is quite challenging due to the unknown, highly nonlinear relation between the two sets of parameters.

The problems encountered with OBD This set of experiments consisted of generating artificial formations, computing the corresponding instrument read-

[1] All values reported in this section are averages over 10 runs.

Table 2: Comparative Results for the Electrical Log Problem.

Training Algorithm	Training Set MSE	Test Set MSE
Back-Propagation		
(10 Hidden Units)	0.0724	0.212
(20 Hidden Units)	0.0718	0.197
(40 Hidden Units)	0.0704	0.209
OBD		
(10 Hidden Units)	0.0717	0.176
(20 Hidden Units)	0.0710	0.287
(40 Hidden Units)	0.0707	0.225
Sparse Connectivity		
(10 Hidden Units)	0.0763	0.222
(20 Hidden Units)	0.0827	0.175
(40 Hidden Units)	0.0724	0.196

ings using a forward model, and training the network to reproduce the actual formation resistivities. Then a second set of artificial formations was generated and the instrument potentials computed. This second set was used as the test set to analyze whether the network had learned the highly non-linear mapping of the backward model.

The training set consisted of 343 vectors, each comprising seven inputs and three outputs. The output resistivities ranged between 1 and 100 ohm–m. The test set consisted of 343 patterns with outputs ranging between 1.5 and 150 ohm–m. The values used for training and testing consisted of the logarithms of the actual values scaled to be between -1 and 1, and the the hyperbolic tangent activation function was used. Table 2 presents the results of the tested algorithms on the test set.

while approximating the Gabor function were also apparent during this set of experiments, and a satisfactory range for the saliencies was difficult to find. Even for a "good" set of parameters, OBD did not provide significant gains over the basic back–propagation, and was in fact outperformed by the sparsely connected network all but once. The sparsely connected network provided a slight improvement over the standard back–propagation algorithm for all but the smallest network, where the number of weights was not large enough to successfully decouple the outputs. These improvements were mainly due to the increased dimensionality of the output space. The sparsely connected network's principle is based on reducing the coupling among the outputs, which is achieved by reducing the number of connections between the hidden layer and the output layer. The performance of such an algorithm improves when the number of output units is increased. It is important to note that the improvements, though not drastic, were obtained without any sacrifice in computational complexity.

5.2 Comparative Studies with a Classification Problem

The data set used for the experiments consists of 25-dimensional feature vectors extracted from short-duration passive sonar signals due to six types of naturally occurring oceanic sources. The signal source type and number of training and test samples are given in Table 3 [46]. Each feature vector consisted of 16 coefficients of Gabor wavelets—a multiscale representation that does not assume signal stationarity; 1 value denoting signal duration, and 8 other temporal descriptors and spectral measurements.

Table 3: Description of the Oceanic Data Set.

Class	Description	Training	Testing
1	Porpoise Sound	116	284
2	Ice 1	116	175
3	Ice 2	78	39
4	Whale Sound 1	116	129
5	Whale Sound 2	148	251
6	Background Noise	116	127
Total:		690	1005

Figures 10–12 show the learning curves for the back–propagation algorithm, the sparsely connected network, and OBD algorithm respectively. Both the basic back–propagation algorithm and the sparsely connected network learn with a smooth gradient with improvements in generalization ability slowing considerably after the initial 150 iterations. The fluctuations in Figure 12 are due to the elimination of weights. In certain cases, the drop in classification accuracy is followed by a subsequent increase overcoming the initial drop (e.g. network with 20 hidden units). In other cases though, the OBD network never reaches the initial level of accuracy (e.g. network with 40 hidden units).

The results obtained by different networks for the classification problem are shown in Table 4. The differences between methods appears to be minimal. Training results indicate that the classification accuracy can be pushed to the limit, without improving the generalization ability. The sparsely connected network was the most consistent, although not the best classifier. Surprisingly, OBD out-performed the back–propagation algorithm for smaller network sizes (10 and 20 hidden units) when theory would suggest improvements taking place with larger networks. This may be due to the increasing difficulty in tuning the thresholds with larger network size. The OBD results for networks with 40 and 80 hidden units may be improved upon if a better threshold could be found.

6 CONCLUDING REMARKS

Even though supervised feedforward networks are often used as model-free black boxes that will learn from examples, effective generalization is contingent on

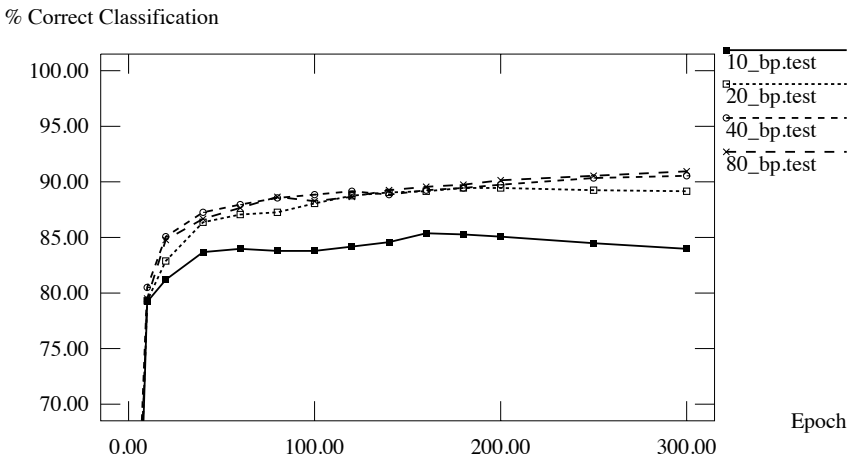

Figure 10: Performance of the Back–propagation algorithm for SONAR classification.

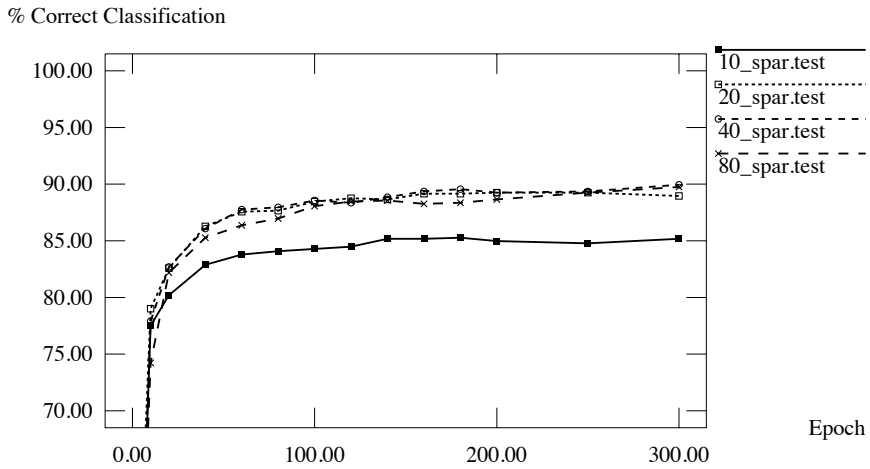

Figure 11: Performance of the Sparsely connected network algorithm for SONAR classification.

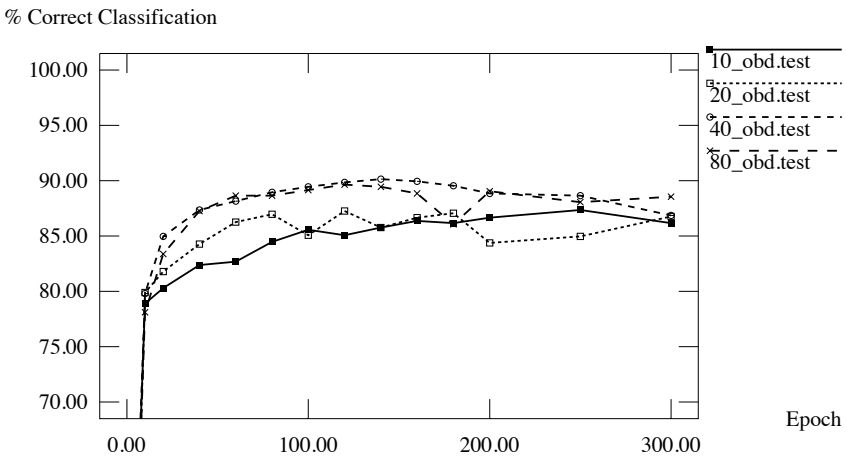

Figure 12: Performance of the OBD algorithm for SONAR classification.

appropriate selection of network structure and the amount of training. We surveyed several methods for dynamically determining the network size and pointed out several theoretical approaches that enable us to understand network behavior. In particular, the concept of complexity regularization provides a powerful framework for developing dynamic-structured networks as well as for estimating performance on the test set using these networks. Heuristics such as weight decay can be well understood within the framework of regularization theory. Bayesian back–propagation prescribes one way of obtaining a suitable amount of regularization during training.

The benefits provided by dynamic architectures have to be weighed against the increased complexity of obtaining these structures. Their performance is sensitive to proper choice of the added parameters, e.g. the cutoff threshold for saliency in OBD, or the value of regularization factor λ used. Heuristics also creep into the choice of lengths for sub-procedures, e.g. the number of epochs needed to train the input layers versus the number of epochs needed to train the output layer in the cascade-correlation algorithm. Indeed, without specific guidelines, the additional time spent in fine-tuning the extra parameters may outweigh the advantages gained by a dynamic algorithm.

Several experimental results indicate that dynamic networks provide only an incremental improvement of the solutions [43] over a static approach. In such cases, further improvement may be obtainable by encoding *a priori* information into the network: either as a set of initial weights, or as the architecture of the network. For example, Kolen and Goel [47] show that the success of the tic-tac-toe player designed by the PDP group [6] is based on the internal representation reflecting a priori knowledge provided to the network. Experiments using a single hidden layer with various number of hidden units, showed that

Table 4: Comparative Results for Oceanic Data Classification.

Training Algorithm	Training Set Correct Classification Rate	Test Set Correct Classification Rate
Back-Propagation		
10 Hidden Units	93.48	83.98
20 Hidden Units	98.55	89.15
40 Hidden Units	98.11	90.54
80 Hidden Units	99.27	90.95
Sparse Connectivity		
10 Hidden Units	90.87	85.17
20 Hidden Units	96.52	89.35
40 Hidden Units	98.84	89.95
80 Hidden Units	98.70	89.75
OBD		
10 Hidden Units	90.14	86.16
20 Hidden Units	92.31	89.26
40 Hidden Units	96.23	90.14
80 Hidden Units	90.43	89.65

the network with random initial weights performed considerably worse than the original experiment where hidden units were designed to reflect certain "bits" of knowledge [47].

When a priori knowledge is not available or easily encodable, the sparsely connected network introduced in this paper can be considered as its generalization ability is better than that of fully connected networks for problems with multi-dimensional output spaces and/or noisy input data. Moreover, these modest improvements are obtained without a sacrifice in either architectural complexity or training time.

This paper concentrated on MLP-type feed-forward networks. While such models are remarkably resilient and applicable over a wide range of approximation problems, it is often worthwhile to investigate whether a specific problem is more amenable to other methods such as spline approximation, projection pursuit, or local potential functions [33, 35], since, as Jerome Friedman reminds us, "there is no method that works best for all problems" [48]. Finding powerful ways to use a priori information in the selection of the *model/network* as well as the training data continues to be an active area of research.

Acknowledgements: This research was supported in part by AFOSR contract F49620-93-1-0307, ONR contract N00014-92-C-0232, and NSF grant ECS 9307632.

References

[1] S. M. Weiss and C.A. Kulikowski. *Computer Systems That Learn*. Morgan Kaufmann, 1991.

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

[3] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.

[4] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[5] A. R. Barron. Complexity regularization with application to neural networks. In G. Roussas, editor, *Nonparametric Functional Estimation and Related Topics*, pages 561–576. Kluwer Academic Press, 1992.

[6] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Bradford Books/MIT Press, Cambridge, Mass, 1986.

[7] B. Widrow and S.D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, Englewood Cliffs, N.J., 1985.

[8] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.

[9] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, Mass, 1969.

[10] Y. LeCun, B. Boser, J.S. Denker, R.E. Henderson, D. Howard, W. Hubbard, and L.D. Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems-2*, pages 396–404. Morgan Kaufmann, 1990.

[11] J.E. Moody. Prediction risk and architecture selection for neural networks. In V. Cherkassky, J.H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks, Proc. NATO/ASI Workshop*, pages 143–156. Springer Verlag, 1994.

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

[13] Y. LeCun, J.S. Denker, and S.A. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems-2*, pages 598–605. Morgan Kaufmann, 1990.

[14] B. Hassibi, D.G. Stork, and G.J. Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, volume 1, pages 293–299, San Fransisco, 1993.

[15] G.E. Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth Annual Conference of the Cognitive Science Society (Amherst 1986)*, pages 1–12, 1986.

[16] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems-4*, pages 950–957. Morgan Kaufmann, 1992.

[17] S. J. Hanson and L. Y. Pratt. Some comparisons of constraints for minimal network construction with back propagation. In D. Touretzky, editor, *Advances in Neural Information Processing Systems-1*. Morgan Kaufmann, 1989.

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

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

[20] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. Touretzky, editor, *Advances in Neural Information Processing Systems-2*, pages 525–532. Morgan Kaufmann, 1990.

[21] E. Littmann and H. Ritter. Generalization abilities of cascade network architecture. In S.J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems-5*, pages 188–195. Morgan Kaufmann, 1993.

[22] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(4):364–378, October 1971.

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

[24] Y. Shin and J. Ghosh. Approximation of multivariate functions using ridge polynomials. In *Proceedings of the International Joint Conference on Neural Networks, Baltimore*, pages II: 380–85, 1992.

[25] T. D. Sanger. A tree-structured adaptive network for function approximation in high-dimensional spaces. *IEEE Trans. on Neural Networks*, 2(2), March 1991.

[26] M. Wynne-Jones. Node splitting: A constructive algorithm for feed-forward neural networks. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems-4*, pages 1072–1079. Morgan Kaufmann, 1992.

[27] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[28] E.J. Hartman, J.D. Keeler, and J.M. Kowalski. Layered neural networks with gaussian hidden units as universal approximators. *Neural Computation*, 2:210–215, 1990.

[29] J. Park and I.W. Sandberg. Universal approximation using radial basis function networks. *Neural Computation*, 3(2):246–257, Summer 1991.

[30] John C. Platt. A resource allocation network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[31] Sukhan Lee and Rhee M. Kil. A gaussian potential function network with hierarchical self-organizing learning. *Neural Networks*, 4:207–224, 1991.

[32] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. (2nd Ed.), Academic Press, 1990.

[33] V. Cherkassky, J.H. Friedman, and H. Wechsler (Eds.). *From Statistics to Neural Networks, Proc. NATO/ASI Workshop*. Springer-Verlag, 1995.

[34] H. White. Learning in artificial neural networks: A statistical perspective. *Neural Computation*, 1(4):425–464, 1989.

[35] R. L. Eubank. *Spline Smoothing and Nonparametric Regression*. M. Dekker, New York, 1988.

[36] G. G. Lorentz. *Approximation of Functions*. Chelsea Publishing, New York, 1986.

[37] T. Poggio and F. Girosi. Networks for approximation and learning. *Proc. IEEE*, 78(9):1481–97, Sept 1990.

[38] G. Cybenko. Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[39] S. Raudys and A.K. Jain. Small sample size problems in designing artificial neural networks. In I.K. Sethi and A. Jain, editors, *Artificial Neural Networks and Statistical Pattern Recognition*, pages 33–50. Elsevier Science, Amsterdam, 1991.

[40] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function theory. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.

[41] W. L. Buntine and A. S. Weigend. Bayesian back-propagation. *Complex Systems*, 5:603–643, 1991.

[42] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, May 1992.

[43] V. Vapnik. Principles of risk minimization for learning theory. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems-4*, pages 831–838. Morgan Kaufmann, 1992.

[44] V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16:264–280, 1971.

[45] J. Ghosh and A. C. Bovik. Processing of textured images using neural networks. In I.K. Sethi and A. Jain, editors, *Artificial Neural Networks and Statistical Pattern Recognition*, pages 133–154. Elsevier Science, Amsterdam, 1991.

[46] J. Ghosh, S. Beck, and C.C. Chu. Evidence combination techniques for robust classification of short-duration oceanic signals. In *SPIE Conf. on Adaptive and Learning Systems, SPIE Proc. Vol. 1706*, pages 266–276, Orlando, Fl., April 1992.

[47] J. F. Kolen and A. K. Goel. Learning in parallel distributed processing networks: Computational complexity and information content. *IEEE Transactions On Systems, Man, and Cybernetics*, 21(2):359–367, March/April 1991.

[48] J. H. Friedman. An overview of nonparametric regression. In *(invited talk) From Statistics to Neural Networks, NATO/ASI Workshop (also see[6])*, Les Arcs, France, 1993.