Comparison of Adaptive Methods for Function Estimation from Samples

Vladimir Cherkassky, *Senior Member, IEEE,* Don Gehring, and Filip Mulier

Abstract— **The problem of estimating an unknown function from a finite number of noisy data points has fundamental importance for many applications. This problem has been studied in statistics, applied math, engineering, artificial intelligence, and, more recently, in the fields of artificial neural networks, fuzzy systems, and genetic optimization. In spite of many papers describing individual methods, very little is known about the comparative predictive (generalization) performance of various methods. We discuss subjective and objective factors contributing to the difficult problem of meaningful comparisons. We also describe a pragmatic framework for comparisons between various methods, and present a detailed comparison study comprising several thousand individual experiments. Our approach to comparisons is biased toward general (nonexpert) users who do not have detailed knowledge of the methods used. Our study uses six representative methods described using a common taxonomy. Comparisons performed on artificial data sets provide some insights on applicability of various methods. No single method proved to be the best, since a method's performance depends significantly on the type of the target function (being estimated), and on the properties of training data (i.e., the number of samples, amount of noise, etc.). Hence, our conclusions contradict many known comparison studies (performed by experts) that usually show performance superiority of a single method (promoted by experts). We also observed the difference in a method's robustness, i.e., the variation in predictive performance caused by the (small) changes in the training data. In particular, statistical methods using greedy (and fast) optimization procedures tend to be less robust than neural-network methods using iterative (slow) optimization for parameter (weight) estimation.**

I. Introduction

In the last decade, neural networks have given rise to high expectations for model-free statistical estimation from a finite number of samples (examples). There is, however, increasing awareness that artificial neural networks (ANN's) represent inherently statistical techniques subject to well-known statistical limitations [1], [2]. Whereas many early neural network application studies have been mostly empirical, more recent research successfully applies statistical notions (such as overfitting, resampling, bias-variance trade-off, the curse of dimensionality, etc.) to improve neural-network performance. Statisticians can also gain much by viewing neural-network methods as new tools for data analysis.

The goal of predictive learning [3] is to estimate/learn an unknown functional mapping between the input (explanatory, predictor) variables and the output (response) variables, from a training set of known (input–output) samples. The mapping is typically implemented as a computational procedure (in software). Once the mapping is obtained/inferred from the training data, it can be used for predicting the output values given only the values of the input variables. Inputs and outputs can be continuous and/or categorical variables. When outputs are continuous variables, the problem is known as regression or function estimation; when outputs are categorical (class labels), the problem is known as classification. There is a close connection between regression and classification in the sense that any classification problem can be reduced to (multiple output) regression [3]. Here we consider only regression problems with a single (scalar) output, i.e., we seek to estimate a function f of $N-1$ predictor variables (denoted by vector $\boldsymbol{x}$) from a given set of n training data points, or measurements, $\mathbf{z}_i = (\mathbf{x}_i, y_i)$, $(i = 1, \cdots, n)$ in N-dimensional sample space

$$y = f(\mathbf{x}) + \text{error} \tag{1}$$

where error is unknown (but zero mean) and its distribution may depend on $\mathbf{x}$. The distribution of training data in $\mathbf{x}$ is also unknown and can be arbitrary.

Nonparametric methods make no or very few general assumptions about the unknown function $f(\mathbf{x})$. Nonparametric regression from finite training data is an ill-posed problem and meaningful predictions are possible only for sufficiently smooth functions. We emphasize that the function smoothness is measured with respect to sampling density of the training data. Additional complications arise due to inherent sparseness of high-dimensional training data (known as the curse of dimensionality) and the difficulty in distinguishing between signal and error terms in (1).

Recently, many adaptive computational methods for function estimation have been proposed independently in statistics, machine learning, pattern recognition, fuzzy systems, nonlinear systems (chaos) and ANN's. General lack of communication between different fields combined with highly specialized terminology often results in duplication or close similarity between methods. For example, there is a close similarity between tree-based methods in statistics (CART) and machine learning (ID3); multilayer perceptron networks use a functional representation similar to projection pursuit regression (PPR) [4], Breiman's PI-method [5] is related to sigma-pi networks [6] as both seek an output in the sum-of-products form, etc. Unfortunately, the problem is not limited to the field-specific jargon, since each field develops its methodology based on its own set of

Manuscript received October 6, 1994; revised May 28, 1995 and January 4, 1996. This work was supported in part by the 3M Corporation.

The authors are with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455 USA.

Publisher Item Identifier S 1045-9227(96)04398-6.

implicit assumptions and modeling goals. Commonly used goals of modeling include: prediction (generalization); explanation/interpretation of the model; usability of the model; biological plausibility of the method; data/dimensionality reduction, etc.

Such diversity makes meaningful comparisons difficult. Moreover, adaptive methods for predictive learning require custom/manual control (parameter tuning) by expert users [7], [8]. Even in a mature field (such as statistics), comparisons are usually controversial, due to the inherent complexity of adaptive methods and the subjective choice of data sets used to illustrate a method's relative performance, i.e., see discussions in [9]. Also, performance comparisons usually do not separate a method from its software implementation. It would be more accurate to discuss comparisons between software implementations rather than methods. Another (less-obvious) implementation bias is due to a method's (implicit) requirements for computational speed and resources. For example, adaptive methods developed by statisticians (MARS, projection pursuit) were (originally) intended for the statistical community. Hence, they had to be fast to be useful for statisticians accustomed to fast methods such as linear regression. In contrast, neural-network implementations were developed by engineers and computer scientists who are more familiar with compute-intensive applications. As a result, statistical methods tend to use fast, greedy optimization techniques, whereas neural-network implementations use more brute force optimization techniques (such as gradient descent, simulated annealing, and genetic algorithms).

II. A Framework for Comparison

Based on the above discussion, meaningful comparisons require:

1) Careful specification of comparison goals, methodology, and design of experiments.
2) Use of fully or semiautomatic modeling methods by nonexpert users. Note that the only way to separate the power of the method from the expertise of a person applying it is to make the method fully automatic (no parameter tuning) or semiautomatic (only a few parameters tuned by a user). Under this approach, automatic methods can be widely used by nonexpert users. The issue is whether adaptive methods can be made semiautomatic without much compromise on their predictive performance. Our experience shows that it can be accomplished by relying on (compute-intensive) internal optimization, rather than user expertise, to choose proper parameter settings.

Specific assumptions used in our comparison study are detailed next.

Goals of Comparison Modeling: Our main criterion is the predictive performance (generalization) of various methods applied by nonexpert users. The comparison does not take into account a methods' explanation/interpretation capabilities, computational (training) time, methods' complexity, etc. All methods (their implementations) should be easy to use, so only minimal user knowledge of the methods is assumed. Training (model inference from data) is assumed off-line and computer time is assigned negligible cost.

Artificial Versus Real-Life Data Sets: Most comparison studies focus on real-life applications. In such studies the main goal is to achieve best results on a given application data set, rather than to provide meaningful comparison of a methods' performance. Moreover, comparison results are greatly affected by application domain-specific knowledge, which can be used for appropriate data encoding/preprocessing and the choice of a method itself. In our experience, such domain-specific knowledge is far more important for successful applications of predictive learning than the choice of a method itself. In addition, application data sets are fixed, and their properties are unknown. Hence, it is generally difficult to evaluate how a methods' performance is affected by the properties of training data, such as sample size, amount of noise, etc. In contrast, characteristics of artificial data are known and can be readily changed. Therefore, from a methodological perspective, we advocate using artificial data sets for comparisons. Of course, any conclusions based on artificial data can be applied to real-life data only if they have similar properties. Characterization of application data remains an important (open) research problem.

Comparison Methodology: The following generic scheme for application of an adaptive method to predictive learning was used:

1) choose a flexible method/representation, i.e., a family of nonlinear (parametric) models indexed by a complexity parameter;
2) estimate/learn model parameters;
3) choose complexity (regularization) parameter of a method (model selection);
4) evaluate predictive performance of the final model.

Note: Steps 2 and 3 may not be distinct in some methods, i.e., early stopping rules in backpropagation training.

Each of the steps 2–4 above generally uses its own data set known as:

- training set, used to estimate model parameters in step 2;
- validation set, used for choosing the complexity parameter of a method in step 3 (i.e., the number of hidden units in a feedforward neural network);
- test set, for evaluating predictive performance of the final model in step 4.

These independent data sets can be readily generated in comparison studies using artificial data. In application studies, when the available data is scarce, the test and validation data can be obtained from the available (training) data via resampling (cross-validation). Sometimes, the terms validation and test data are used interchangeably, since many studies use the same (test) samples to choose (optimally) the complexity parameter and to evaluate the predictive performance of the final model [8], [10]. In our study, we also used the same samples (test set) in steps 3 and 4.

Taking the validation set to be the same as the test set also avoids the problem of model selection in our study, as explained next. Empirically observed predictive performance

depends on the outcome of all steps (1)–(3). Hence, comparisons between methods may be complicated by the choice of the complexity term/parameter in Step (3), also known as model selection, a very difficult problem by itself. Methods for choosing a complexity parameter is an area of active research, and they include data resampling (cross-validation) and analytic methods for estimating model prediction error [11], [12]. The problem of model selection is avoided in our comparisons to:

1) focus on the comparison of a methods' representation power and optimization/estimation capabilities (that can be obscured by the results of model selection);
2) avoid computational cost of resampling techniques for model selection. Instead, we choose to spend computational resources on comparing a method's performance on many different data sets.

Software Package XTAL for Nonexpert Users: To enable/improve usability of adaptive methods by nonexpert users, several statistical and neural-network methods for nonparametric regression (developed elsewhere) were integrated into a single package XTAL (stands for crystal) with a uniform user interface (common for all methods). Note that any large-scale comparison involving hundreds or thousands experiments would not be practically feasible with methods implemented as stand alone modules. Thus, XTAL incorporates a sequencer that allows it to cycle through large number of experiments without operator intervention. In addition, all methods were modified so that, at most, one or two parameters need to be user-defined (no limit is imposed on internal parameter tuning transparent to a user). Since most adaptive methods in the package originally had a large number of user-tunable parameters (typically half a dozen or so), most of these parameters were either set to carefully chosen default values or internally optimized in the final version included in the package. Naturally, the final choice of user-tunable parameters and the default values is somewhat subjective and it introduces a certain bias into comparisons between methods. This is the price to pay for the simplicity of using adaptive methods under our approach. The XTAL package was developed by the authors who had no detailed knowledge of every method incorporated into the package.

III. Taxonomy of Methods for Function Estimation

It is important to provide a common taxonomy of statistical and neural network methods for function estimation to interpret the results of empirical comparisons. Reasonable taxonomies can be based on a method's

1) representation scheme for the target function (being estimated);
2) optimization strategy;
3) interpretation capability.

In this paper, we follow the representation-scheme taxonomy where the function is estimated as a linear combination of basis functions (basis function expansion) [3]

$$\hat{f}(\mathbf{x}) = \sum_{j=0}^{M} a_j B_j(\mathbf{x}, \mathbf{p}_j) \tag{2}$$

where

1) $\mathbf{x}$ is a vector of input variables;
2) a_j are expansion coefficients (to be determined from data);
3) $B(\mathbf{x}, \mathbf{p})$ are basis functions;
4) $\mathbf{p}_j$ are parameters of each basis function;
5) usually $B(\mathbf{x}, \mathbf{p}_0) = 1$;
6) M is the regularization parameter of a method.

Methods based on this taxonomy are also known as dictionary methods [3], since the methods differ according to the set of the basis functions (or dictionary) they use. We can further distinguish between nonadaptive (parametric) and adaptive methods, as follows:

1) Nonadaptive methods use preset basis functions (and their parameters), so that only coefficients a_j are fit to data. Optimal values for a_j are (usually) found by least squares from n training samples, by minimizing

$$\sum_{i=1}^{n} \left[y_i - \sum_{j=0}^{M} a_j B_j(\mathbf{x}_i, \mathbf{p}_j) \right]^2. \tag{3}$$

There are two major classes of nonadaptive methods, i.e., global parametric methods (such as linear and polynomial regression), and local parametric methods (such as kernel smoothers, piecewise-linear regression and splines). For a good discussion of nonadaptive methods, see [9]. Note that global parametric methods inevitably introduce bias and local parametric methods are applicable only to low-dimensional problems (due to inherent sparseness of finite samples in high-dimensions known as "the curse of dimensionality"). Hence, adaptive methods are the only practical alternative for high-dimensional problems.

2) Adaptive methods, where (in addition to coefficients a_j basis functions themselves and/or their parameters $\mathbf{p}_j$ are adapted to data. For adaptive methods optimization (3) becomes a difficult (nonlinear) problem. Hence, the optimization strategy used becomes very important. Statistical methods usually adopt greedy optimization strategy (stepwise selection) where each basis function is estimated one at a time. In contrast, neural-network methods usually optimize over the whole set of basis functions.

In this paper, we are mostly concerned with adaptive methods. All reasonable adaptive methods use a set of basis functions rich enough to provide universal approximation, i.e., for all target functions $f(\mathbf{x})$ of some specified smoothness and for any $e > 0$ there exist M, a_j, $\mathbf{p}_j (j = 1, \cdots, M)$ such that

$$\left\| f(\mathbf{x}) - \sum_{j=0}^{M} a_j B_j(\mathbf{x}, \mathbf{p}_j) \right\| < e. \tag{4}$$

What is a good choice for the basis functions (method) used? In general, it depends on the (unknown) target function being estimated, in the sense that the best dictionary (method) is the one that provides the "simplest" representation in the form (2) for a given

(prespecified) accuracy of estimation. The simplest-form representation, however, does not mean just the number of terms in (2), since different methods may use basis functions of different complexity. For example, ANN's use fixed (sigmoid) univariate basis functions of linear combinations (projections) of input variables, whereas PPR uses arbitrary univariate basis functions of projections. Since PPR uses more complex basis functions than ANN's, its estimates would generally have fewer terms in (2) than those of ANN.

Adaptive methods can be further classified as:

1) Global methods, which use basis functions globally defined in the domain of $\mathbf{x}$. The most popular choice are univariate functions of projections (linear combinations) of the input variables, as in ANN's and PPR. This choice is very attractive since it automatically achieves dimensionality reduction. Other methods for choosing global basis functions are known in statistics, such as additive models [14], tensor-product splines in MARS [9], sum-of-products basis functions used in PI-method [5], etc.
2) Local methods, that use local basis functions (in $\mathbf{x}$-space). Such methods either use local basis functions explicitly (such as radial basis function networks with locally-tuned units) or implicitly via adaptive distance metric in $\mathbf{x}$-space (as in adaptive-metric nearest neighbors, adaptive kernel smoothers, partitioning methods such as CART, etc.). Such methods effectively perform data-adaptive local feature selection.

IV. Description of Representative Methods

Based on the taxonomy of methods presented above, a number of regression methods have been selected for comparison and included in the XTAL package. These methods were chosen to represent a member of each of the major classes of methods. Each method [except generalized memory-based learning (GMBL)] is available as public-domain software developed by its original author(s). Next we describe these methods and their parameter settings.

A. Nearest Neighbors

A simple version of k-nearest neighbors regression was implemented in the XTAL package a benchmark. Nearest neighbors is a locally parametric method where the response value for a given input is an average of the k closest training samples (in $\boldsymbol{x}$-space) to this input. The value of k controls the amount of smoothing performed and is set by the user in the XTAL package.

B. Generalized Memory-Based Learning

GMBL [15], [16] is a statistical technique that was designed specifically for robotic control. The model is based on storing past samples of training data to "learn by example." When new data arrives, an output is determined by interpolating. GMBL is capable of using either weighted nearest neighbor or locally weighted linear interpolation (loess) [17]. The interpolating method and parameters used are a critical part of the model. They must be chosen very carefully so that overfitting does not occur. GMBL uses cross-validation to select the smoothing parameters, the distance scale used for each variable and method with the best fit. The method's parameters are adjusted to minimize the cross-validation estimate using optimization techniques. This parameter selection is very time consuming and is done off-line. After the parameter selection is completed, the power of the method is in its capability to perform prediction with data as it arrives in real-time. It also has the ability to deal with nonstationary processes by "forgetting" past data. Since the GMBL model depends on weighted nearest neighbor or locally weighted linear interpolation, its estimates are rather similar to (but usually more accurate than) the results of a naive k-nearest neighbors regression. GMBL performs well in low-dimensional cases, but high-dimensional data sets make parameter selection critical and very time-consuming. Original GMBL code provided by Moore [16] was used. The GMBL version in the package has no user-defined parameters. Default values of the original GMBL implementation were used for the internal model selection.

C. Projection Pursuit

Projection pursuit [4] is a global adaptive method which exhibits good performance in high-dimensional problems and is invariant to linear coordinate transformations. The model generated by this method is the sum of univariate functions g_i of linear combinations of the elements of $\mathbf{x}$

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^{M} g_j(\mathbf{p}_j^T \mathbf{x}).$$

The parameters $\mathbf{p}_j$ and the functions g_j are adaptively optimized based on the data. For each of the M projections, the algorithm determines the best $\mathbf{p}_j$ using a gradient descent technique to search for the projection which minimizes the unexplained variance. Each g_j is a smoothed version of the projected data with smoothing parameters chosen according to a fit criteria such as cross-validation. Since the model is additive, the search for function projections is done iteratively using the so-called backfitting algorithm. This is a greedy optimization technique where each additive term is estimated one at a time. The model is decomposed based on unexplained variance

$$y - \sum_{\substack{j=1 \\ j \neq k}}^{M} g_j(\mathbf{p}_j^T \mathbf{x}) = g_k(\mathbf{p}_k^T \mathbf{x}).$$

The $\mathbf{p}_k$ is optimally chosen using gradient descent while holding the $\mathbf{p}_j$, $j \neq k$ fixed. In each iteration another term is pulled out of the summation and an optimal $\mathbf{p}_k$ is found. This procedure is repeated until the average residual does not vary significantly. In this way, each function projection is chosen to best fit the largest unexplained variance of the data.

Theoretically it is possible to model any smooth function with projection pursuit for large enough M [18]. For large M, however, the approximation is computationally time consuming and difficult to interpret. The method approximates radial function well, but harmonic functions are better approximated by kernel estimators [19]. PP also exhibits a sensitivity to

outliers, since their presence increases the chance of choosing a spurious projection. This occurs because the search for projections can get caught in a local minima. In terms of speed of execution, the method is limited by the speed of the smoother and the rate of convergence of the optimizing algorithm. In the original implementation of PP [13], the supersmoother is employed for smoothing. Other implementations of projection pursuit have used Hermite polynomials [20]. In general, a very robust adaptive smoother is required, which may cause speed limitations.

The original implementation of projection pursuit, called SMART (smooth multiple additive regression technique) [13], employs a heuristic search strategy for selecting the number of projections to avoid poor solutions due to multiple local minima. The SMART user must select the largest number of projections (M_L) to use in the search as well as the final number of projections (M_F). The strategy is to start with M_L projections and remove projections based on their relative importance until the model has M_F projections. The model with M_F projections is then returned as the regression solution. To improve ease of use in the XTAL package, M_F is set by the user, but M_L is always taken to be $M_F + 5$. In addition, the SMART package allows the user to control the thoroughness of optimization. In the XTAL implementation, this was set to the highest level.

D. Artificial Neural Networks (ANN's)

Multilayer perceptrons with a single hidden layer and a linear output unit compute a linear combination of basis functions (2), where the basis functions are fixed (sigmoid) univariate functions of linear combinations of input variables. This is a global adaptive method in our taxonomy. Various training (learning) procedures for such networks differ primarily in the optimization strategy used to estimate parameters (weights) of a network. The XTAL package uses a version of multilayer feedforward networks with a single hidden layer described in [21]. This version employs conjugate gradient descent for estimating model parameters (weights) and performs a very thorough (internal) optimization via simulated annealing to escape from local minima (10 annealing cycles). The original implementation from [21] was used with minor modifications. The method's implementation in XTAL has a single user-defined parameter—the number of hidden units. This is the complexity parameter of the method. There is a close similarity between PPR and ANN's in terms of representation, as both methods use nonlinear univariate basis functions of linear combinations (projections) of input variables. The two methods use very different optimization procedures, however: PPR uses greedy (stepwise) optimization to estimate additive terms in (2) one at a time, whereas ANN training estimates all the basis functions simultaneously.

E. Multivariate Adaptive Regression Splines

MARS [9] is a global adaptive method in our taxonomy. This method combines the idea of recursive partitioning regression (CART) [22] with function representation based on tensor-product splines. The method of recursive partitioning consists of adaptively splitting the sample space into disjoint regions and modeling each region with a constant value. The regions are chosen based on a greedy optimization procedure where in each step, the algorithm selects the split which causes the largest decrease in mean squared error. A basis function for each region can be described by

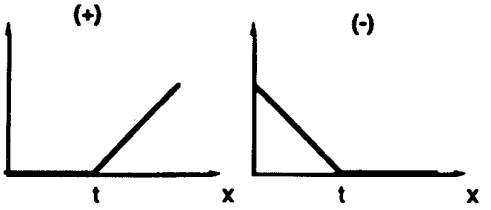

Fig. 1. Pair of one-dimensional basis functions used by the MARS method.

$$B_j(\mathbf{x}) = I[\mathbf{x} \in R_j] \tag{5}$$

where I is the indicator function. In this case, I has the value one if the vector $\mathbf{x}$ is in region R_j and zero otherwise. The model can then be described by the following expansion on these basis functions

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^{M} a_j B_j(\mathbf{x}). \tag{6}$$

The MARS method is based on similar principles of recursive partitioning and greedy optimization, but uses continuous basis functions rather than ones based on the indicator function. The basis functions of the MARS algorithm can each be described in terms of a two-sided truncated power basis function (truncated spline) of the form

$$B_q^{\pm}(x - t) = [\pm(x - t)]_+^q \tag{7}$$

where t is the location of the knot, q is the order (of splines) and the + subscript denotes the positive part of the argument. The basic building block of the MARS model is a pair of these basis functions which can be adjusted using coefficients to give a local approximation to data (Fig. 1). For multivariate problems, products of the univariate basis functions are used. The basis functions for MARS can be described by

$$B_j^{(q)}(x) = \prod_{k=1}^{K_j} [s_{j,k} \cdot (\mathbf{x}_{v(j,k)} - t_{j,k})]_+^q. \tag{8}$$

This is a product of one-dimensional splines each with a directional term ($s_{j,k} = \pm 1$). The variable K_j defines the number of splits required to define the region j, v indicates the particular variable of $\mathbf{x}$ used in the splitting and $t_{j,k}$ is the split point.

The MARS model can be interpreted as a tree where each node in the tree consists of a basis function and uses a tree-based algorithm for constructing the model. Like other recursive partitioning methods, nodes are split according to a goodness of fit measure. MARS differs from other partitioning methods in that all nodes (not just the leaves) of the tree are candidates for splitting. Fig. 2 shows an example of a MARS tree. The function described is

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^{7} a_j B_j(\mathbf{x}). \tag{9}$$

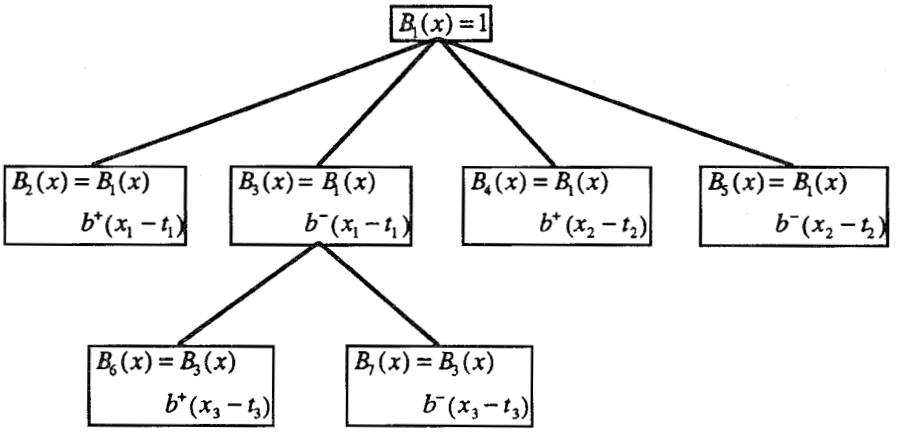

Fig. 2. Example of a MARS tree.

The depth of the tree indicates the interaction level. On each path down variables are split at most once. The algorithm for constructing the tree uses a forward stepwise and backward stepwise strategy. In the forward stepwise procedure, a search is performed over every node in the tree to find a node, which when split improves the fit according to the fit criteria. This search is done over all candidate variables, split points $t_{j,k}$, and basis coefficients. For example, in Fig. 2 the root node $B_1(\mathbf{x})$ is split first on variable x_1, and the two daughter nodes $B_2(\mathbf{x})$ and $B_3(\mathbf{x})$ are created. Then, the root node is split again on variable x_2 creating the nodes $B_4(\mathbf{x})$ and $B_5(\mathbf{x})$. Finally node $B_3(\mathbf{x})$ is split on variable x_3. In the backward stepwise procedure, leaves are removed which cause either an improved fit or slight degradation in fit as long as model complexity decreases.

The measure of fit used by the MARS algorithm is the generalized cross validation (GCV) estimate [23]. This GCV measure provides an estimate of the future prediction accuracy and is determined by measuring the mean squared error on the training set and penalizing this measurement to account for the increase of variance due to model complexity. The user can select the amount of penalization imposed (in terms of degrees of freedom) for each additional split used by the algorithm. Theoretical and empirical studies seem to indicate that adaptive knot location adds between two and four additional model parameters for each split [9]. The user also selects the maximum number of basis functions and the interaction degree for the MARS algorithm. In the XTAL implementation, the user selects the maximum number of basis functions and the degrees of freedom (recoded as an integer from zero to nine). The interaction degree is defaulted to allow all interactions.

The MARS method is well suited for high- as well as low-dimensional problems with a small number of low-order interactions. An interaction occurs when the effect of one variable depends on the level of one or more other variables and the order of the interaction indicates the number of interacting variables. Like other recursive partitioning methods, MARS is not robust in the case of outliers in the training data. It also has the disadvantage of being sensitive to coordinate rotations. For this reason, the performance of the MARS algorithm is dependent on the coordinate system used to represent the data. This occurs because MARS partitions the space into axis-oriented subregions. The method does have some advantages in terms of speed of execution, interpretation, and relatively automatic smoothing parameter selection.

F. Constrained Topological Mapping (CTM)

CTM [24] is an example of local adaptive method. In terms of representation of the regression estimate, CTM is similar to CART, where the input ($\mathbf{x}$) space is partitioned into disjoint (unequal) regions, each having a constant response (output) value. Unlike CART's greedy tree partitioning, however, CTM uses (nonrecursive) partitioning strategy originating from the neural network model of self-organizing maps (SOM's) [25]. In the CTM model, a (high-dimensional) regression surface is estimated (represented) using a fixed number of "units" (or local prototypes) arranged into a (low-dimensional) topological map. Each unit has ($\mathbf{x}$, y), coordinates associated with it, and the goal of training (self-organization) is to position the map units to achieve faithful approximation of the unknown function. The CTM model uses a suitable modification (for regression problems) of the original SOM algorithm [25] to faithfully approximate the unknown regression surface from the training samples. The main modification is that the best-matching unit step of SOM algorithm is performed in the space of predictor variables ($\mathbf{x}$-space), rather than in the full ($\mathbf{x}$, y) sample space [24]. The effectiveness of SOM/CTM methods in modeling high-dimensional distributions/functions is due to the use of low-dimensional maps. The use of topological maps effectively results in performing kernel smoothing in the (low-dimensional) map space, which constitutes a new approach to dimensionality reduction and dealing with the curse of dimensionality [26]. In the regression problem, one assumes data of the form $\mathbf{z}_k = (\mathbf{x}_k, y_k)$, $(k = 1, \cdots, K)$. Effectively, the CTM algorithm performs Kohonen self-organization in the space of the predictor variables $\mathbf{x}$ and performs an additional update to determine a piecewise constant regression estimate of y for each unit. In this algorithm the unit locations are

denoted by the vectors $\mathbf{w_j}$ where $\mathbf{j}$ is the vector topological coordinate for the unit. Units of the map are first initialized uniformly along the principal components of the data. Then, the following three iteration steps are used to update the units:

1) Partitioning: Bin the data according to the index of the nearest unit based on the predictor variables $\mathbf{x}$. In this step, the data are recoded so that each vector data sample is associated with the topological coordinates of the unit nearest to it in the predictor space.

$$\mathbf{i}_k = \underset{j}{\text{argmin}} \; ||\mathbf{x}_k - \mathbf{w_j}|| \text{ for each data vector, } \mathbf{x}_k, \; (k = 1, \cdots, K).$$

2) Conditional expectation estimate: Determine the new unit locations based on nonparametric regression estimates using the recoded data as in the SOM algorithm. The original SOM algorithm essentially used kernel smoothing to estimate the conditional expectations, where the kernel was given by the neighborhood function in the topological space. Additionally, update the response estimates for each unit. Treat the topological coordinates found above, $\mathbf{i}_k$ as the predictors and the $\mathbf{x}_k$ as a vector response. New unit locations are given by the regression estimates at all topological coordinate locations

$$\mathbf{w_j} = \frac{\sum_{k=1}^{K} \mathbf{x}_k H_x(||\mathbf{i}_k - \mathbf{j}||)}{\sum_{k=1}^{K} H_x(||\mathbf{i}_k - \mathbf{j}||)}, \text{ this is an estimate of } E(\mathbf{x}|\mathbf{j})$$

$$\hat{y}_j = \frac{\sum_{k=1}^{K} y_k H_y(||\mathbf{i}_k - \mathbf{j}||)}{\sum_{k=1}^{K} H_y(||\mathbf{i}_k - \mathbf{j}||)}, \text{ this is an estimate of } E(y|\mathbf{j})$$

for all valid topological coordinates $\mathbf{j}$. Here H_x is the kernel function used for updating unit locations and H_y is the kernel function used for updating the response estimates.

3) Neighborhood decrease: The complexity of each of the estimates is independently increased. When using kernel smoothing, this corresponds to decreasing the span of the smoother.

The trained CTM provides piecewise-constant interpolation between the units. The constant-response regions are defined in terms of the Voronoi regions of the units in the predictor space. Prediction based on this model is essentially a table lookup. For a given input $\mathbf{x}$, the nearest unit is found in the space of the predictor variables and the piecewise constant estimate for that unit is given as response.

Empirical results [24], [27], [28] have shown that the original CTM algorithm provides accurate regression estimates. It lacks, however, some key features found in other statistical methods:

1) Piecewise-linear versus piecewise-constant approximation: The original CTM algorithm uses a piecewise constant regression surface, which is not an accurate representation scheme for smooth functions. Better accuracy could be achieved using, for example, a piecewise-linear fit.
2) Control of model complexity: Up to this point, there has been little understanding of how model complexity in the CTM algorithm is adjusted. Interpreting the unit update equations as a kernel regression estimate [26] gives some insight by interpreting the neighborhood width as a kernel span in the topological map space. The neighborhood decrease schedule then plays a key role in the control of complexity.
3) Global variable selection: Global variable selection is a popular statistical technique commonly used (in linear regression) to reduce the number of predictor variables by discarding low-importance variables. The original CTM algorithm, however, provides no information about variable importance, since it gives all variables equal strength in the clustering step. Since the CTM algorithm performs self-organization (clustering) based on Euclidean distance in the space of the predictor variables, the method is sensitive to predictor scaling. Hence, variable selection can be implemented in CTM indirectly via adaptive scaling of predictor variables during training. This scaling makes the method adaptive, since the quality of the fit in the response variable affects the positioning of map units in the predictor space. This feature is important for high dimensional problems where training samples are sparse, since local parametric methods require dense samples and global parametric methods introduce bias. Hence, adaptive methods are the only practical alternative.
4) Batch versus flow-through implementation. The original CTM (as most neural-network methods) is a flow-through algorithm, where samples are processed one at a time. Even though flow-through methods may be desirable in some applications (i.e., control), they are generally inferior to batch methods (that use all available training samples) commonly used in statistics for estimation, both in terms of computational speed and estimation accuracy. In particular, the results of modeling using flow-through methods may depend on the (heuristic) choice of the learning rate schedule [29]. Hence, the batch version of CTM has been developed in [30].

These deficiencies in the original CTM algorithm can be overcome using statistically-motivated improvements, as detailed next. These improvements have been incorporated in the latest version of CTM included in XTAL package.

1) Local Linear Regression Estimates: The original CTM algorithm, can be modified to provide piecewise linear approximation for the regression surface. Using locally weighted multiple linear regression, the neighborhood function would be used to weight the observations and zero and first-order

terms can be estimated for each unit. The regression estimate for each unit is global, but local samples are given more weight according to the neighborhood function. This differs from the flow-through procedure proposed by Ritter *et al.* [28] which effectively uses linear regression over each Voronoi region separately, and then forms a weighted average of the regression coefficients using the neighborhood function to determine the first-order estimate for each unit. This method does not take into account the density of points in each Voronoi region, since the coefficient averaging process is done over the regions independent of the number of samples falling in each region. Such averaging also makes it difficult to see what error functional is being minimized and is statistically inefficient. The method proposed here is similar to loess [17], in that a weighted mean-squared error criterion is minimized. It differs from loess in that the neighborhood function rather than a nearest neighbor ranking is used to weight the observations. Using a piecewise linear regression surface rather than piecewise constant gives CTM more flexibility in function fitting. Hence, fewer units are required to give the same level of accuracy. Also, the limiting case of a map with one unit corresponds to linear regression. The regression surface produced by CTM using linear fitting is not guaranteed to be continuous at the edges of the Voronoi regions. The neighborhoods of adjacent units overlap, however, so the linear estimates for each region are based on common data samples. This imposes a mild constraint which tends to induce continuity.

2) Using Cross-Validation to Select Final Neighborhood Width: The complexity of a model produced by the SOM or CTM method is determined by the final neighborhood width used in the training algorithm [26]. In other words, the final neighborhood width is a smoothing (regularization) parameter of the CTM method. To estimate the correct amount of smoothing from the data, one commonly uses cross-validation. In this procedure, a series of regression estimates are determined based on portions of the training data, and the sum of squares error is measured using the remaining validation samples. For each regression estimate, a different subset of validation samples are chosen from the original training set, so that each training sample is used exactly once for validation. The final measure of generalization error is the average of the sum of squares error. Because of the computational advantages, leave-one-out cross validation was chosen for CTM. In this case, each sample is systematically removed from the training set to be used as the validation set. If the regression estimation procedure can be decomposed as a linear matrix operation on the training data, then the leave one out cross validation score can be easily computed [14].

3) Variable Selection via Adaptive Sensitivity Scaling: The CTM algorithm effectively applies the Kohonen SOM in the space of the predictor variables to determine the unit locations. The SOM is essentially a clustering technique, which is sensitive to the particular distance scaling used. For CTM, a heuristic scaling technique can be implemented based on the sensitivity of the linear fits for each Voronoi region. We would like to adjust the scales of the predictor variables so that those variables that are most important in the regression are given more weight in the distance calculation. The sensitivity of a variable on the regression surface can be determined locally for each Voronoi region. These local sensitivities can be averaged over the Voronoi regions to judge the global importance of a variable on the whole regression estimate. Since new regression estimates are given with each iteration of the CTM algorithm, this scaling can be done adaptively, i.e., the scaling/variable importance parameters are used in the distance calculation when clustering the data according to the Voronoi regions. This effectively causes more units to be placed along variable axis which have larger average sensitivity.

4) Implementation Details: To make a regression method practical, there are a number of implementation issues that must be addressed. The Batch CTM software package provides some additional features that improve the quality of the results and improve the ease of use. For example, in interpolation (no noise) problems with a small number of samples, model selection based on cross-validation provides an overly smooth estimate. For these problems it may be advantageous to do model selection based on the mean squared error on the training set. The package allows the user to select either cross-validation or training set error or a mixture of the two to perform model selection. This effectively provides the user control over a complexity penalty. The package is also capable of automatically estimating the number of units of the map based on the error (cross-validation or training set) score. This heuristic procedure provides good automatic selection of the number of units when the user does not wish to enter a specific number. When used with XTAL, the user supplies the model complexity penalty, an integer from 0 to 9 (max. smoothing) and the dimensionality of the map.

V. Experimental Setup

Experiment design includes the specification of:

1) types of functions (mappings) used to generate samples;
2) properties of the training and test data sets;
3) specification of performance metric used for comparisons;
4) description of modeling methods used (including default parameter settings).

Functions Used: In the first part of our study, artificial data sets were generated for eight "representative" two-variable functions taken from statistical and ANN literature. These include different types of functions, such as harmonic, additive, complicated interaction, etc. In the second part of comparisons, several high-dimensional data sets were used, i.e., one six-variable function and four-variable functions. These high-dimensional functions include intrinsically low-dimensional functions that can be easily estimated from data, as well as difficult functions for which model-free estimation (from limited-size training data) is not possible. The list of all 13 functions is shown in Appendix I.

Training Data Characteristics Include Its Distribution, Size, and Noise: Training set distribution was uniform in x-space. Since using a random number generator to create a small number of samples results in somewhat clustered samples

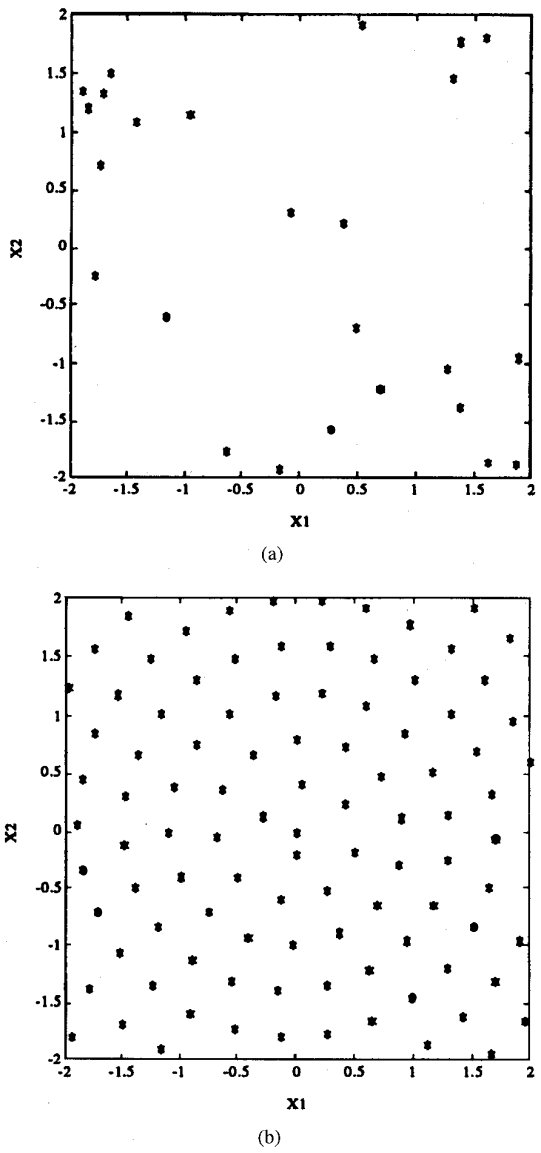

Fig. 3. (a) 25 samples from a uniform distribution. (b) Uniform spiral distribution with 100 samples.

[see Fig. 3(a)], a uniform spiral distribution was also used to produce more uniform samples for the two-variable functions [see Fig. 3(b)]. The random distribution was used for high-dimensional functions 9–12. The spiral distribution was used for function 13 to generate the values of two hidden variables that were transformed into four-dimensional training data.

The main motivation for introducing the uniform spiral distribution was to eliminate the variability in model estimates due the variance of finite random samples in x-space [as shown in Fig. 3(a)], without resorting to averaging of the model estimates over many random samples. The usual averaging approach does not seem practically feasible, given the size of this study (several thousand individual experiments). The solution to this dilemma taken in this study was to run two sequences of experiments for each of the two-dimensional functions; one sequence was run using random distributions and another, otherwise identical, sequence was run using spiral distributions. These spiral distributions were created by placing samples at evenly spaced points along a linear spiral in such a way that the samples were always of even density throughout the surface of space. Thus, the spiral distribution is the polar equivalent of a uniform rectangular grid based on Cartesian coordinates, but it has the advantage that its points do not lie on lines parallel to Cartesian axes. The uniform spiral distribution corresponds to the designed experiment setting as opposed to observational setting (that favors random distributions).

Training set size: Three sizes were used for each function and distribution type (random and uniform spiral), i.e., small (25 samples), medium (100 samples), and large (400 samples).

Training set noise: The training samples were corrupted by three different levels of Gaussian noise: no noise, medium noise (SNR = 4), and high noise (SNR = 2). Thus there were a total of 189 training data sets generated, i.e., eight two-variable functions with two distribution types, three sample sizes, and three noise levels ($8*2*3*3 = 144$), and five high-dimensional functions with a single distribution type, three sample sizes, and three noise levels ($5 * 1 * 3 * 3 = 45$).

Test Data: A single data set was generated for each of the thirteen functions used. For two-variable functions, the test set had 961 points uniformly spaced on a 31 × 31 square grid. For high-dimensional functions, the test data consisted of 961 points randomly sampled in the domain of X.

Performance Metric: The performance index used to compare predictive performance (generalization capability) of the methods is the normalized root mean square error (NRMS), i.e., the average RMS on the test set normalized by the standard deviation of the test set. This measure represents the fraction of unexplained standard deviation. Hence, a small value of NRMS indicates good predictive performance, whereas a large value indicates poor performance (the value NRMS = 1 corresponds to the "mean response" model).

Modeling Methods: Methods included in the XTAL package were described in Section IV.

User-Controlled Parameter Settings: Each method (except GMBL) was run four times on every training data set, with the following parameter settings:

1) KNN: k = 2, 4, 8, 16.
2) GMBL: no parameters (run only once).
3) CTM: map dimensionality set to 2, smoothing parameter = 0, 2, 5, 9.
4) MARS: 100 maximum basis functions, smoothing parameter (degrees-of-freedom) = 0, 2, 5, 9.
5) PPR: number of terms (in the smallest model) = 1, 2, 5, 8.
6) ANN: number of hidden units = 5, 10, 20, 40.

Number of Experiments: With 189 training data sets and six modeling methods, each applied with four parameter settings (except GMBL applied once), the total number of experiments performed is: $189 * 5 * 4 + 189 * 1 = 3969$. Most other comparison studies on regression typically use only tens of experiments. The sheer number of experimental data reveals many interesting insights that (sometimes) contradict the findings from smaller studies.

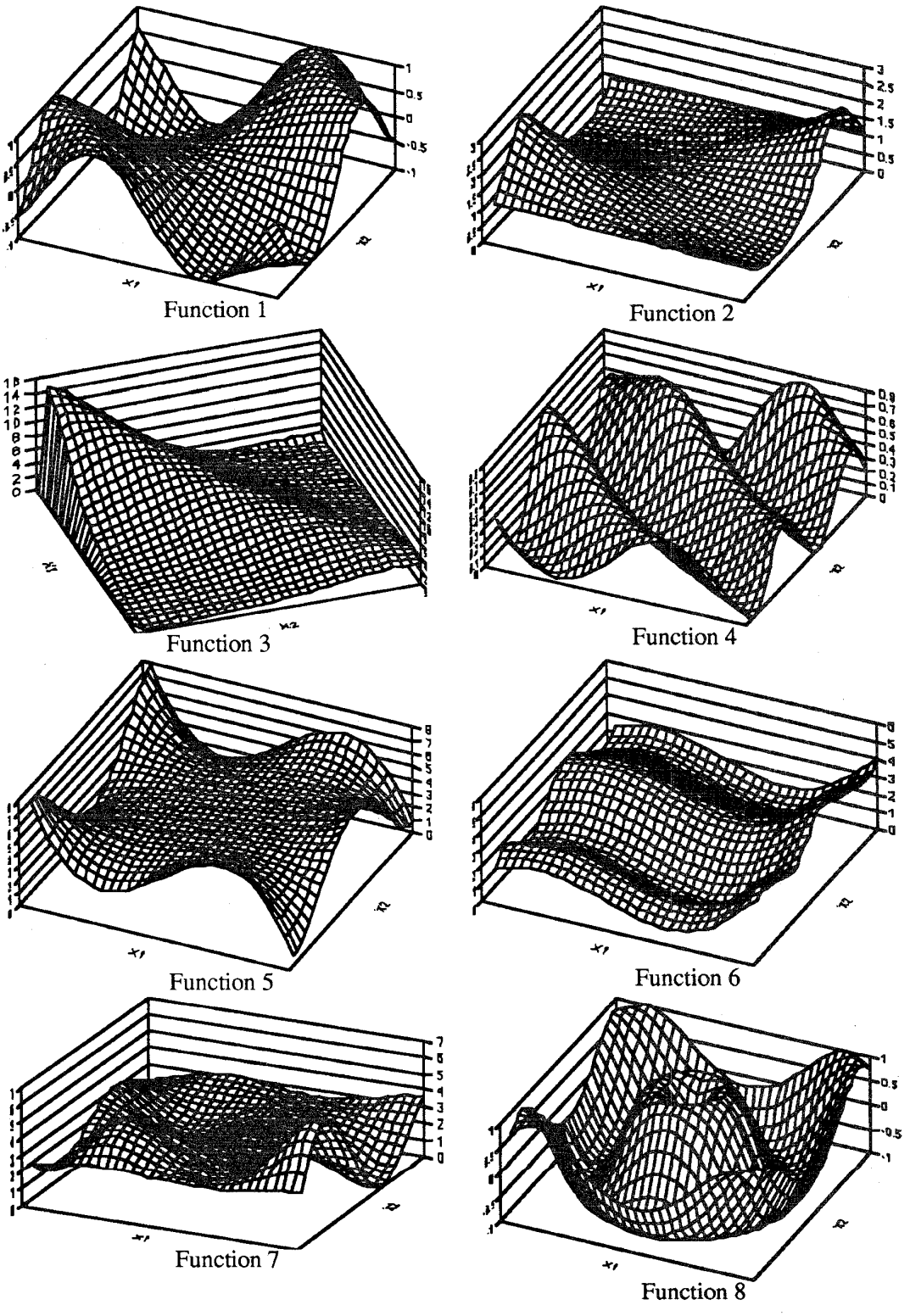

Fig. 4. Representations of the two variable functions used in the comparisons.

VI. Discussion of Experimental Results

Experimental results summarizing nearly 4000 individual experiments are presented in Appendix II. Each table shows the comparative performance of methods for a particular target function. For each method, four user-controlled parameter values were tried, and only the best model (with the smallest NRMS on the test set) was used for compar-

isons. Then the best methods were marked as crosses in a comparison table. Often, methods showed very close best performance (within 5%); hence several winners (crosses) are entered in the table row. The absolute prediction performance is indicated by the best NRMS error value in the left column (this value corresponds to the best method and is marked with bold cross). For two-variable functions, the two NRMS values in each row correspond to the random and spiral distribution of training samples, respectively. For small samples and/or difficult (high-dimensional) functions, often none of the methods provided a model better than the response mean (i.e., NRMS ≥ 1). In such cases, there were no winners, and no entries were made in the comparison tables.

Each method's performance is discussed next with respect to:

1) type of function (mapping);
2) characteristics of the training set, i.e., sample size/distribution and the amount of added noise;
3) a method's robustness with respect to characteristics of training data and tunable parameters. Robust methods show small (predictable) variation in their predictive performance in response to small changes in the (properties of) training data or tunable parameters (of a method). Methods exhibiting robust behavior are preferable for two reasons: first, they are easier to tune for optimal performance, and second, their performance is more predictable and reliable.

K-NN and GMBL Observations: These two local methods provide qualitatively similar performance, and their predictions are usually inferior in situations where more accurate model estimation (by other, more structured methods) is possible. K-NN and GMBL, however, perform best in situations where accurate estimation is impossible as indicated by high normalized RMS values (i.e., NRMS > 0.75). This can be seen most clearly in situations with small sample size and/or nonsmooth functions, i.e., functions 11 and 12 (the four-dimensional "multiplicative" and "cascaded" functions) where other methods were completely unable to produce usable results (i.e., NRMS value > 1 indicating that estimation error was greater than the standard deviation for the test set).

Both methods utilize a memory based structure which makes it possible to add new training samples without having to retrain the method. This can be an important advantage in some situations because a method such as ANN can take many hours to retrain.

Overall, both K-NN and GMBL showed very predictable (robust) behavior with respect to changes in sample size, noise levels, and function class which made their results dependable performance measures. All other methods showed far greater variability, particularly with respect to function class.

CTM Observations: In this study CTM did well at estimating harmonic functions (functions 1, 5, and 8). This result is consistent with those of Cherkassky, Lee, and Lari-Najafi [27] which studied functions 5, 6, and 7 and that of Cherkassky and Mulier [30] which included function 8. This result is not surprising since CTM is similar to kernel methods which are known to work best with harmonic functions. CTM performs rather poorly on functions of linear combinations of input variables (i.e., 4 and 6). Overall, CTM exhibits robust behavior, except at small samples. On a wide range of function types CTM displayed a peculiar ability to give exceptionally good results in interpolation situations where the sample size was large (400) and there was no added noise.

MARS Observations: For two-variable functions, MARS performed best with "additive" function 6. It also performed best when estimating the high-dimensional "additive" functions (9 and 10). In fact, function 10 is much more linear than its analytical form suggests, due to the small range of its input variables. It can be accurately approximated by Taylor series expansion around $X = 0$, which lends itself to linear representation. MARS performed poorly when used with two-dimensional data sets of 25 samples.

PPR Observations: PPR's performance was similar to, but generally worse than, that of ANN. Both methods use a common representation, i.e., the sum of functions of linear combinations of input variables. Thus, performance of these two methods is well suited to a function like four which can be reduced to this form. On the two-dimensional "additive" function (6) the performance of projection pursuit was superior to all other methods tested. As noted by Maechler *et al.* [10] harmonic functions are a worst case for projection pursuit. This is evident in this study by the particularly poor performance of projection pursuit on the harmonic functions 5 and 8.

Projection pursuit is highly sensitive to the correct choice of it's "term" parameter. This parameter is, however, difficult to choose and best results can only be obtained by testing a large number of different values. It is this facet of projection pursuit that probably best accounts for the significant improvement in the results reported in this study over those previously reported in Maechler *et al.* [10] and Hwang *et al.* [20].

ANN Observations: For most of the two-dimensional functions (functions 1, 2, 3, 4, 5, 7, and 8) and the four-dimensional function with the underlying two-dimensional function (function 13), ANN gave the best overall estimation performance. This was evident both in the score for total wins as well as the score based on a winner take all basis. ANN was robust with respect to sample size, noise level, and choice of its tuning parameter (number of hidden units). ANN was outperformed on additive functions by projection pursuit in two dimensions (function 6) and by MARS in high-dimensional functions (functions 9 and 10). The excellent performance of ANN can be explained by its ability to approximate three common types of functions: 1) functions of linear combinations of input variables; 2) radial functions, due to the observation that ANN and PP use a similar representation, and in view of the known theoretical results on the suitability of PP to approximate (asymptotically optimally) radial functions [19]; and 3) harmonic functions, since a kernel function can be constructed as a linear combination of sigmoids [10], and kernel methods are known to approximate harmonic functions well [19].

The performance of ANN was generally somewhat better than that reported in previous comparisons studied by Maechler *et al.* [10], Hwang *et al.* [20], and Cherkassky *et al.* [27]. We attribute this difference to the simulated annealing algorithm used in this study to escape from local minima during training. This optimization is computationally expensive, and results in training times in excess of several hours on the Sun 4 workstations used. Run times for ANN were at least one, and typically two, orders of magnitude greater than for other methods.

General Comments on All Methods: Overall, most methods provide comparable predictive performance for large samples. This is not surprising, since all (reasonable) adaptive methods are asymptotically optimal (universal approximators). A methods' performance becomes more uneven at smaller sample sizes. The comparative performance of different methods is summarized below:

	BEST	**WORST**
Prediction accuracy (dense samples)	ANN	KNN, GMBL
Prediction accuracy (sparse samples)	GMBL, KNN	MARS, PP
Additive target functions	MARS, PP	KNN, GMBL
Harmonic functions	CTM, ANN	PP
Radial functions	ANN, PP	KNN
Robustness wrt parameter tuning	ANN, GMBL	PP
Robustness wrt sample properties	ANN, GMBL	PP, MARS.

Here denseness of samples is measured with respect to the target function complexity (i.e., smoothness), so that in our study dense sample observations refer mostly to medium/large sample sizes for two-variable functions, and sparse sample observations refer to small sample results for two-variable functions as well as all sample sizes for high-dimensional functions.

The small number of high-dimensional target functions included in this comparison study makes any definite conclusions difficult. Our results, however, confirm the well-known notion that high-dimensional (sparse) data can be effectively estimated only if its target function has some special property. For example, additive target functions (9 and 10) can be accurately estimated by MARS; whereas functions with correlated input variables (function 13) can be accurately estimated by ANN, GMBL, and CTM. On the other hand, examples of inherently complex target functions (11 and 12) can not be accurately estimated by any method due to sparseness of training data. An interesting observation is that whenever accurate estimation is not possible (i.e., sparse samples), more structured methods generally fail, but memory-based methods provide better accuracy.

Methods' Robustness: Even though all methods in our study are flexible nonparametric function estimators, there are two unstructured nonparametric methods (GMBL and KNN), whereas the other methods (ANN, PP, MARS, and CTM) introduce certain structure (implicit assumptions) into estimation. For example, the ANN method assumes that unknown function can be accurately estimated by a large number of simple sigmoid functions, whereas PP assumes that a function can be estimated by a small number of complex univariate functions of linear combinations of input variables. Our results indicate that unstructured methods (such as GMBL and KNN) are generally more robust than other (more structured) methods. Of course, better robustness does not imply better prediction performance.

Also, neural-network methods (ANN, CTM) are more robust than statistical ones (MARS, PP). This is due to differences in the optimization procedures used. Specifically, greedy (stepwise) optimization commonly used in statistical methods results in more brittle model estimates than the neural network-style optimization, where all the basis functions are estimated together in an iterative fashion.

Comparison with Earlier Studies: Since our comparison is biased toward (and performed by) nonexpert users, and the number of user-tunable parameters and their values (for all methods) was intentionally limited to only four parameter settings, the quality of the reported results may be suspect. It is entirely possible that the same adaptive methods can provide more accurate estimates if they are applied by expert users who can tune their parameters at will. To show that our approach provides (almost) optimal results, we present comparisons with earlier studies performed by the experts using the same (or very similar) experimental setup. The comparison tables for three two-variable functions (harmonic, additive and complicated interaction) originally used in [10], are shown in Appendix III. All results from previous studies are rescaled to the same performance index (NRMS on the test set) used in this study. The first table in Appendix III shows NRMS error comparisons for several methods using 400 training samples. Original results for ANN, PP, and CTM were given in [10] and [24], whereas original MARS results were provided by Friedman [9]. In all cases (except one) this study provides better NRMS than the original studies. The difference could be explained as follows:

1) in the case of ANN, by a more thorough optimization (via simulated annealing) in our version;
2) in the case of PP, by better choice of default parameter values in our version;
3) in the case of MARS, by a different choice of user-tunable parameter values;
4) in the case of CTM, our study employs a new batch version with piecewise-linear approximation [31], whereas the original study used a flow-through version with piecewise-constant approximation.

The remaining three tables in Appendix III show comparisons on the same three functions, but using 225 samples with no noise added and with added noise. The original study [20] uses the same SMART implementation of PP (as in our study), but a different version of ANN. Since our study does not use 225 samples, the tables show the results for 100 and 400 samples for comparison. It would be reasonable to expect that the results for 225 samples (of the original study) should fall in between 225 and 400 samples of our study. In fact, in most cases our results for 100 samples are better than the results for 225 samples from the original study. This provides an additional confidence in our results, as well as

an empirical evidence in favor of the premise that complex adaptive methods can be automated without compromise on their prediction accuracy.

VII. Conclusions

We discussed the important but intrinsically difficult subject of comparisons between adaptive methods for predictive learning. Many subjective and objective factors contributing to the problem of meaningful comparisons were enumerated and discussed. A pragmatic framework for comparisons of various methods by nonexpert users was presented. This approach is proved feasible by developing a software package XTAL that incorporates several adaptive methods for regression for nonexpert users, and successfully applying this software to perform a massive comparison study comprising thousands of individual experiments. As expected, none of the methods provided superior prediction accuracy for all situations. Moreover, a method's performance was found to depend significantly and nontrivially on the properties of training data. For example, no (single) method was able to give optimal performance, for a given target function, at all combinations of sample size and noise level. Hence, we may conclude that the real value of comparisons lies in the interpretation/explanation of comparison results. This implies the need for a meaningful characterization of the modeling (estimation) methods as well as the data sets. Our paper provides a method's characterization using a general taxonomy of methods for function estimation based on the type of basis functions and on the optimization procedure used.

Experimental results presented in this paper can be used to draw certain conclusions on the applicability of representative methods to various data sets (with well-defined characteristics). For example:

1) additive target functions are best estimated using statistical methods (MARS, PP);
2) projection-based methods (ANN and PP) are the best choice for the (target) functions of linear combinations of input variables;
3) harmonic functions are best estimated using CTM and ANN;
4) neural-network methods tend to be more robust than statistical ones;
5) unstructured methods (nearest neighbor and kernel-smoother type) tend to be more robust than more structured methods.

The problem of choosing a method for a given application data set is a difficult one. A very effective solution to this problem was successfully demonstrated in this project by automating the sequencing of a rather "brute force" comparison of available regression methodologies. A very substantial reduction in the amount of time required to familiarize a novice user with advanced modeling methods is achieved by providing a single universal control scheme that supports all methods and hides unnecessary details from the user. Even for an experienced user, the execution of the long sequences of comprehensive experiments made possible by the software package developed for this project would be completely impossible to accomplish by manual methods.

We emphasize that the results of any comparison (including ours) on a method's predictive performance should be taken with caution. Such comparisons do not (and in most cases, can not) differentiate between methods and their software implementations. In some situations, however, poor relative performance of a method may be due to its software implementation. Such problems caused by software implementation can be usually detected only by expert users, typically by the original authors/implementers of a method. In addition, a method's performance can be adversely affected by the choice (or poor implementation) of the estimation/optimization procedure (i.e., step 3 in the generic method outline given in Section II). For example, the final performance of multilayer networks (ANN) greatly depends on the optimization technique used to estimate model parameters (weights). Similarly, predictive performance of PP depends on the type of smoother used to estimate nonlinear functions of projections [20]. The results of this study may suggest that a brute force compute-intensive approach to optimization pays off (assuming that computing power is cheap), as indicated by the success of empirical methods such as ANN and CTM. Specifically in the case of ANN, this study achieves much better performance than in [10] and [20] on the same data sets, due to the use of simulated annealing to escape from local minima. More research is needed, however, to evaluate and quantify the effect of optimization procedures on predictive performance.

Appendix I
Functions Used to Generate Data Sets

Function 1 from Breiman [1991]
$y = \sin(x_1 * x_2)$ X uniform in [-2, 2]

Function 2 from Breiman [1991]
$y = \exp(x_1 * \sin(\pi * x_2))$ X uniform in [-1, 1]

Function 3 - the GBCW function from Gu et al [1990]
$a = 40 * \exp(8*((x_1 - .5)^2 + (x_2 - .5)^2))$
$b = \exp(8*((x_1 - .2)^2 + (x_2 - .7)^2))$
$c = \exp(8*((x_1 - .7)^2 + (x_2 - .2)^2))$
$y = a/(b + c)$ X uniform in [0, 1]

Function 4 from Masters [1991]
$y = (1 + \sin(2x_1 + 3x_2))/(3.5 + \sin(x_1 - x_2))$ X uniform in [-2, 2]

Function 5 (harmonic) from Maechler et al [1990]
$y = 42.659(2 + x_1)/20 + \mathrm{Re}(z^5))$ where $z = x_1 + ix_2 - .5(1 + i)$

or equivalently with $x_1 = x_1 - .5$, $x_2 = x_2 - .5$
$y = 42.659(.1 + x_1(.05 + x_1^4 - 10x_1^2x_2^2 + 5x_2^4))$ X uniform in [-.5, .5]

Function 6 (additive) from Maechler et al [1990]
$y = 1.3356[1.5(1 - x_1) + \exp(2x_1 - 1)\sin(3\pi(x_1 - .6)^2) + \exp(3(x_2 - .5))\sin(4\pi(x_2 - .9)^2)]$ X uniform in [0,1]

Function 7 (complicated interaction) from Maechler et al [1990]
$y = 1.9[1.35 + \exp(x_1)\sin(13(x_1 - .6)^2)\exp(-x_2)\sin(7x_2)]$ X uniform in [0, 1]

Function 8 (harmonic) from Cherkassky et al [1991]
$y = \sin(2\pi * \mathrm{sqrt}(x_1^2 + x_2^2))$ X uniform in [-1, 1]

Function 9 (6-dimensional additive) adapted from Friedman [1991]
$y = 10\sin(\pi x_1 x_2) + 20(x_3 - .5)^2 + 10x_4 + 5x_5 + 0x_6$ X uniform in [-1, 1]

Function 10 (4-dimensional additive)
$y = \exp(2x_1\sin(\pi x_4)) + \sin(x_2 x_3)$ X uniform in [-.25, .25]

Function 11 (4-dimensional multiplicative)
$y = 4(x_1 - .5)(x_4 - .5)\sin(2\pi\ \mathrm{sqrt}(x_2^2 + x_3^2))$ X uniform [-1, 1]

Function 12 (4-dimensional cascaded)
$a = \exp(2x_1\sin(\pi x_4))$, $b = \exp(2x_2\sin(\pi x_3))$
$y = \sin(a * b)$ X uniform in [-1, 1]

Function 13 (4 nominal variables, 2 hidden variables)
$y = \sin(a * b)$
where hidden variables a, b are from uniform spiral in [-2, 2].
Observed (nominal) X-variables are: $x_1 = a*\cos(b)$, $x_2 = \mathrm{sqrt}(a^2 + b^2)$, $x_3 = a + b$, $x_4 = a$

APPENDIX II
EXPERIMENTAL RESULTS

Function 1 $y = \sin(x_1 * x_2)$

	RANDOM DISTRIBUTION						SPIRAL DISTRIBUTION					
	KNN	GMBL	CTM	MARS	PP	ANN	KNN	GMBL	CTM	MARS	PP	ANN
small # samples												
no noise .440/.314	X	X			X	X		X	X			X
medium .517/.390						X		X				X
high .660/.534	X	X						X				
medium # samples												
no noise .110/.061				X		X			X	X		X
medium .205/.210					X	X		X	X		X	X
high .369/.315	X	X	X	X	X	X			X		X	X
large # samples												
no noise .033/.017			X	X		X			X	X		X
medium .095/.098			X	X		X		X	X	X	X	X
high .182/.125		X			X	X		X	X		X	X
WINS	3	4	3	4	4	8	0	6	7	3	4	8
W.T.A.	1	2	0	1	0	5	0	1	2	0	0	6

Function 2 $y = \exp(x_1 * \sin(\pi * x_2))$

	RANDOM DISTRIBUTION						SPIRAL DISTRIBUTION					
	KNN	GMBL	CTM	MARS	PP	ANN	KNN	GMBL	CTM	MARS	PP	ANN
small # samples												
no noise .375/.253						X					X	X
medium .511/.328					X	X					X	X
high .622/.572						X		X			X	
medium # samples												
no noise .117/.060				X	X	X			X	X		X
medium .167/.154				X		X				X	X	X
high .266/.261						X		X				X
large # samples												
no noise .016/.015			X	X	X				X	X	X	
medium .098/.100		X	X	X		X		X	X	X	X	X
high .136/.146						X			X			X
WINS	0	1	2	4	3	8	0	3	4	4	6	7
W.T.A.	0	0	0	1	1	7	0	0	1	1	2	5

Function 3

$a = 40 * \exp(8*((x_1 - .5)^2 + (x_2 - .5)^2))$
$b = \exp(8*((x_1 - .2)^2 + (x_2 - .7)^2))$
$c = \exp(8*((x_1 - .7)^2 + (x_2 - .2)^2))$
$y = a/(b + c)$

	RANDOM DISTRIBUTION						SPIRAL DISTRIBUTION					
	KNN	GMBL	CTM	MARS	PP	ANN	KNN	GMBL	CTM	MARS	PP	ANN
small # samples												
no noise .092/.050						X						X
medium .343/.212		X						X			X	X
high .414/.451		X						X			X	
medium # samples												
no noise .037/.029					X	X			X		X	X
medium .099/.096						X						X
high .286/.202	X	X			X	X					X	X
large # samples												
no noise .016/.010			X	X	X	X			X	X	X	X
medium .075/.060		X			X	X		X	X		X	X
high .125/.148						X	X	X	X		X	X
WINS	1	4	1	1	4	7	1	4	4	1	7	8
W.T.A.	1	2	0	0	1	5	0	0	1	0	3	5

Function 4 $y = (1 + \sin(2x_1 + 3x_2))/(3.5 + \sin(x_1 - x_2))$

	RANDOM DISTRIBUTION						SPIRAL DISTRIBUTION					
	KNN	GMBL	CTM	MARS	PP	ANN	KNN	GMBL	CTM	MARS	PP	ANN
small # samples												
no noise .419/.359						X						X
medium .843/.626					X	X					X	
high .709/.650					X						X	
medium # samples												
no noise .175/.136						X					X	X
medium .285/.180					X	X						X
high .366/.327					X	X						X
large # samples												
no noise .120/.052			X		X	X			X			
medium .150/.126						X						X
high .198/.220					X	X						X
WINS	0	0	1	0	6	8	0	0	1	0	5	6
W.T.A.	0	0	0	0	3	6	0	0	1	0	2	6

Function 5 (harmonic)

$y = 42.659(2 + x_1)/20 + \mathrm{Re}(z^5))$ where $z = x_1 + ix_2 - .5(1 + i)$

or equivalently with $x_1 = x_1 - .5$, $x_2 = x_2 - .5$

$y = 42.659(.1 + x_1(.05 + x_1^4 - 10x_1^2x_2^2 + 5x_2^4))$

	RANDOM DISTRIBUTION						SPIRAL DISTRIBUTION					
	KNN	GMBL	CTM	MARS	PP	ANN	KNN	GMBL	CTM	MARS	PP	ANN
small # samples												
no noise .821/.614		X						X				X
medium .934/.504						X						X
high .867/.664	X	X						X	X			X
medium # samples												
no noise .225/.187						X			X	X		X
medium .399/.245						X						X
high .422/.347		X				X			X			X
large # samples												
no noise .059/.038			X	X					X	X		
medium .172/.131			X	X		X		X	X	X		X
high .185/.199						X		X	X			X
WINS	1	3	2	2	0	6	0	4	6	3	0	8
W.T.A.	1	1	1	1	0	5	0	1	4	0	0	4

Function 6

$y = 1.3356[1.5(1 - x_1) + \exp(2x_1 - 1)\sin(3\pi(x_1 - .6)^2) + \exp(3(x_2 - .5))\sin(4\pi(x_2 - .9)^2)]$

	RANDOM DISTRIBUTION						SPIRAL DISTRIBUTION					
	KNN	GMBL	CTM	MARS	PP	ANN	KNN	GMBL	CTM	MARS	PP	ANN
small # samples												
no noise .400/.189				X	X	X					X	
medium .360/.250					X						X	
high .921/.450	X										X	
medium # samples												
no noise .029/.030				X	X					X	X	
medium .149/.113					X						X	X
high .321/.240					X	X				X		X
large # samples												
no noise .010/.008				X	X				X	X	X	
medium .068/.065				X	X	X					X	X
high .144/.161				X	X	X					X	X
WINS	1	0	0	5	8	4	0	0	1	3	8	4
W.T.A.	1	0	0	1	6	1	0	0	0	2	5	2

Function 7

$y = 1.9[1.35 + \exp(x_1)\sin(13(x_1 - .6)^2)\exp(-x_2)\sin(7x_2)]$

	RANDOM DISTRIBUTION						SPIRAL DISTRIBUTION					
	KNN	GMBL	CTM	MARS	PP	ANN	KNN	GMBL	CTM	MARS	PP	ANN
small # samples												
no noise .498/.453						X		X				X
medium .493/.474						X		X				X
high .640/.509	X				X			X	X		X	
medium # samples												
no noise .218/.112			X		X	X			X			X
medium .255/.192						X		X				X
high .394/.338		X				X						X
large # samples												
no noise .061/.034			X	X	X				X	X		
medium .142/.125			X		X	X			X	X	X	X
high .238/.148					X	X						X
WINS	1	1	3	1	5	7	0	4	4	2	2	7
W.T.A.	0	0	1	0	1	7	0	2	3	0	0	4

Function 8

$y = \sin(2\pi * \mathrm{sqrt}(x_1^2 + x_2^2))$

	RANDOM DISTRIBUTION						SPIRAL DISTRIBUTION					
	KNN	GMBL	CTM	MARS	PP	ANN	KNN	GMBL	CTM	MARS	PP	ANN
small # samples												
no noise .792/.628		X				X		X			X	
medium .920/.646	X	X			X			X	X			
high /.842								X	X			
medium # samples												
no noise .276/.156						X			X			X
medium .245/.238						X						X
high .436/.347						X						X
large # samples												
no noise .106/.049			X						X			
medium .196/.162			X		X	X			X			X
high .227/.248			X			X			X			X
WINS	1	2	3	0	2	6	0	3	6	0	1	5
W.T.A.	0	0	2	0	1	5	0	3	2	0	0	4

Function 9

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - .5)^2 + 10x_4 + 5x_5 + 0x_6$$

RANDOM DISTRIBUTION

	KNN	GMBL	CTM	MARS	PP	ANN
small # samples						
no noise .496		X		X	X	
medium .546		X	X	X		
high .677	X	X	X	X		
medium # samples						
no noise .099					X	
medium .319				X		X
high .456		X		X		
large # samples						
no noise .039				X	X	
medium .152					X	X
high .269						X
WINS	1	4	2	6	4	3
W.T.A.	0	3	0	2	1	3

Function 10

$$y = \exp(2x_1\sin(\pi x_2)) + \sin(x_3 x_4)$$

RANDOM DISTRIBUTION

	KNN	GMBL	CTM	MARS	PP	ANN
small # samples						
no noise .283				X	X	
medium .272				X		
high .514				X		
medium # samples						
no noise .026				X		
medium .146				X		
high .246			X			
large # samples						
no noise .011				X	X	
medium .107				X		X
high .182				X	X	X
WINS	0	0	1	8	2	2
W.T.A.	0	0	1	7	0	1

Function 11

$$y = 4(x_1 - .5)(x_2 - .5)\sin(2\pi\ \mathrm{sqrt}(x_3^2 + x_4^2))$$

RANDOM DISTRIBUTION

	KNN	GMBL	CTM	MARS	PP	ANN
small # samples						
no noise						
medium						
high						
medium # samples						
no noise .821	X	X				
medium .971	X	X				
high						
large # samples						
no noise .668	X	X				
medium .710	X	X				
high .762	X	X				
WINS	5	5	0	0	0	0
W.T.A.	3	2	0	0	0	0

Function 12 (4-dimensional cascaded)

$a = \exp(2x_1\sin(\pi x_2))$, $b = \exp(2x_3\sin(\pi x_4))$

$y = \sin(a * b)$

RANDOM DISTRIBUTION

	KNN	GMBL	CTM	MARS	PP	ANN
small # samples						
no noise						
medium						
high						
medium # samples						
no noise						
medium						
high						
large # samples						
no noise .890	X	X				
medium .882	X	X				
high .887	X	X				
WINS	3	3	0	0	0	0
W.T.A.	0	3	0	0	0	0

Function 13 (4 nominal variables, 2 hidden variables)

$y = \sin(a * b)$

where hidden variables a, b are from uniform spiral in [-2, 2].

Observed (nominal) X-variables are:

$x_1 = a*\cos(b)$, $x_2 = \mathrm{sqrt}(a^2 + b^2)$, $x_3 = a + b$, $x_4 = a$

RANDOM DISTRIBUTION

	KNN	GMBL	CTM	MARS	PP	ANN
small # samples						
no noise .423		X	X			X
medium .465		X			X	
high .510		X				
medium # samples						
no noise .052			X			X
medium .237		X		X		X
high .308			X		X	X
large # samples						
no noise .012			X	X		X
medium .102		X	X	X	X	X
high .179						X
WINS	0	5	5	3	3	7
W.T.A.	0	3	1	0	0	5

APPENDIX III
COMPARISONS WITH EARLIER STUDIES

Original Study: Maechler et al [1990]; Cherkassky et al [1991]

Experimental set-up

number of training samples: 400
noise level: large (SNR = 4)
sample distribution: uniform grid (earlier studies)
uniform spiral (this study)

	ANN	PP	CTM	MARS
function 5 (harmonic)				
original study	.308	.504	**.131**	.190
this study	**.151**	**.230**	**.131**	**.169**
function 6 (additive)				
original study	.096	.128	.170	**.063**
this study	**.095**	**.065**	**.147**	.122
function 7 (complicated)				
original study	.227	.206	.197	.179
this study	**.126**	**.141**	**.125**	**.138**

Original study: Hwang et at [1994]

Function #5 (harmonic)

SAMPLE SIZE / SN RATIO	GMBL	CTM	MARS	PP	ANN
ORIGINAL STUDY:					
225/no noise	N/A	N/A	N/A	.498	.428
225/SN=4	N/A	N/A	N/A	.504	.457
THIS STUDY:					
100/no noise	.256	**.187**	.199	.383	.206
100/SN=4	.299	.308	.308	.440	**.245**
400/no noise	.149	**.038**	.066	.259	.133
400/SN=4	.154	**.131**	.169	.229	.151

Function #6 (additive)

SAMPLE SIZE / SN RATIO	GMBL	CTM	MARS	PP	ANN
ORIGINAL STUDY:				.022	.057
225/no noise	N/A	N/A	N/A	.136	.141
225/SN=4	N/A	N/A	N/A		
THIS STUDY:					
100/no noise	.196	.224	**.030**	.035	.077
100/SN=4	.244	.300	.187	**.113**	.146
400/no noise	.169	.031	**.008**	.008	.065
400/SN=4	.170	.147	.122	**.065**	.095

Function #7 (complicated interaction)

SAMPLE SIZE / SN RATIO	GMBL	CTM	MARS	PP	ANN
ORIGINAL STUDY:				.168	.146
225/no noise	N/A	N/A	N/A	.208	.265
225/SN=4	N/A	N/A	N/A		
THIS STUDY:					
100/no noise	.182	.138	.243	.156	**.113**
100/SN=4	.237	.399	.242	.246	**.192**
400/no noise	.155	**.034**	.046	.081	.099
400/SN=4	.178	**.125**	.139	.141	.126

ACKNOWLEDGMENT

The authors wish to thank J. H. Friedman of Stanford University for providing PP and MARS code and A. W. Moore of Carnegie Mellon University for providing GMBL code used in this project. Several stimulating discussions with J. H. Friedman on the comparisons between methods are also gratefully acknowledged.

REFERENCES

[1] B. D. Ripley, "Pattern Recognition and Neural Networks," in *Statist. Images.* Cambridge, U.K.: Cambridge University Press, 1996.
[2] V. Cherkassky, J. H. Friedman, and H. Wechsler, Eds., *From Statistics to Neural Networks: Theory and Pattern Recognition Applications,* vol. 136. Berlin: Springer-Verlag, NATO ASI Series F, 1994.
[3] J. H. Friedman, "An overview of predictive learning and function approximation," in *From Statistics to Neural Networks: Theory and Pattern Recognition Applications,* vol. 136, V. Cherkassky, J. H. Friedman, and H. Wechsler, Eds. Berlin: Springer-Verlag, NATO ASI Series F, 1994.
[4] J. H. Friedman and W. Stuetzle, "Projection pursuit regression," *J. Amer. Statist. Assoc.,* vol. 76, pp. 817–823, 1981.
[5] L. Breiman, "The PI method for estimating multivariate functions from noisy data," *Technometrics,* vol. 3, no. 2, pp. 125–160, 1991.
[6] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, Eds., *Parallel Distributed Processing.* Cambridge, MA: MIT Press, 1986.
[7] A. S. Weigend and N. A. Gershenfeld, Eds., *Time Series Prediction: Forecasting the Future and Understanding the Past.* Reading, MA: Addison-Wesley, 1993.
[8] K. Ng and R. P. Lippmann, "A comparative study of the practical characteristics of neural network and conventional pattern classifiers," MIT Lincoln Lab., Tech. Rep. 894, 1991.
[9] J. H. Friedman, "Multivariate adaptive regression splines (with discussion)," *Ann. Statist.,* vol. 19, pp. 1–141, 1991.
[10] M. Maechler, D. Martin, J. Schimert, M. Csoppensky, and J. N. Hwang, "Projection pursuit learning networks for regression," in *Proc. Int. Conf. Tools AI,* 1990, pp. 350–358.
[11] J. Moody, "Prediction risk and architecture selection for neural networks," in *From Statistics to Neural Networks: Theory and Pattern Recognition Applications,* vol. 136, V. Cherkassky, J. H. Friedman, and H. Wechsler, Eds. Berlin: Springer-Verlag, NATO ASI Series F, 1994.
[12] V. N. Vapnik, *The Nature of Statistical Learning Theory.* Berlin: Springer-Verlag, 1995.
[13] J. H. Friedman, "SMART user's guide," Dep. Statistics, Stanford Univ., Tech. Rep. 1, 1984.
[14] T. Hastie and R. Tibshirani, *Generalized Additive Models.* New York: Chapman and Hall, 1990.
[15] C. G. Atkeson, "Memory-based approaches to approximating continuous functions," in *Proc. Wkshp. Nonlinear Modeling Forecasting,* Santa Fe, NM, 1990.
[16] A. W. Moore, "Fast robust adaptive control by learning only feedforward models," in *Advances in NIPS-4J,* E. Moody *et al.,* Eds. San Mateo, CA: Morgan Kaufmann, 1992.
[17] W. S. Cleveland and S. J. Delvin, "Locally weighted regression: An approach to regression analysis by local fitting," *J. Amer. Statist. Assoc.,* vol. 83, no. 403, pp. 596–610, 1988.
[18] P. Diaconis and M. Shahshahani, "On nonlinear functions of linear combinations," *SIAM J. Sci. Statist. Comput.,* vol. 5, pp. 175–191, 1984.
[19] D. L. Donoho and I. M. Johnstone, "Projection-based approximation and a duality with kernel methods," *Ann. Statist.,* vol. 17, no. 1, pp. 58–106, 1989.
[20] J. Hwang, S. Lay, M. Maechler, and R. D. Martin, "Regression modeling in backpropagation and projection pursuit learning," *IEEE Trans. Neural Networks,* vol. 5, pp. 342–353, 1994.
[21] T. Masters, *Practical Neural Network Recipes in C++.* New York: Academic, 1993.
[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees.* Belmont, CA: Wadsworth, 1984.
[23] P. Craven and G. Wahba, "Smoothing noisy data with spline functions," *Numer. Math.,* vol. 31, pp. 377–403, 1979.
[24] V. Cherkassky and H. Lari-Najafi, "Constrained topological mapping for nonparametric regression analysis," *Neural Networks,* vol. 4, pp. 27–40, 1991.
[25] T. Kohonen, *Self-Organization and Associative Memory.* Berlin: Springer-Verlag, 1989.
[26] F. Mulier and V. Cherkassky, "Self-organization as an iterative kernel smoothing process," *Neural Computa.,* vol. 7, no. 6, pp. 1165–1177, 1995.
[27] V. Cherkassky, Y. Lee, and H. Lari-Najafi, "Self-organizing network for regression: Efficient implementation and comparative evaluation," in *Proc. IJCNN,* vol. 1, 1991, pp. 79–84.
[28] H. Ritter, T. Martinetz, and K. Schulten, *Neural Computation and Self-Organizing Maps: An Introduction.* Reading, MA: Addison-Wesley, 1992.
[29] F. Mulier and V. Cherkassky, "Statistical analysis of self-organization," *Neural Networks,* vol. 8, no. 5, pp. 717–727, 1995.
[30] V. Cherkassky and F. Mulier, "Application of self-organizing maps to regression problems," *J. Amer. Statist. Assoc.,* submitted for publication, 1994.

Vladimir Cherkassky (S'83–M'85–SM'92) received the M.S. degree in systems engineering from the Moscow Aviation Institute, Russia, in 1976, and the Ph.D. degree in electrical engineering from the University of Texas at Austin in 1985.

He is Associate Professor of Electrical Engineering at the University of Minnesota, Twin Cities campus. His current research is on theory and applications of neural networks to data analysis, knowledge extraction, and process modeling.

Dr. Cherkassky was actively involved in the organization of several conferences on artificial neural networks. He was Director of the NATO Advanced Study Institute (ASI) From Statistics to Neural Networks: Theory and Pattern Recognition Applications held in France in 1993. He is a member of the program committee and session chair at the World Congress on Neural Networks (WCNN) in 1995 and 1996.

Don Gehring received the B.S. degree summa cum laude in computer science at the University of Minnesota, Twin Cities campus, in 1996.

Previously, while at the University of California at Berkeley, he founded a company providing computer system design services for scientific research. The study reported in this paper was presented as a thesis project for B.S. degree at the University of Minnesota.

Filip Mulier received the B.S. degree, the M.S. degree, and the Ph.D. degree in 1989, 1992, and 1994, respectively, all in electrical engineering at the University of Minnesota, Twin Cities campus. His dissertation work was on the analysis and characterization of self-organizing maps from a statistical viewpoint.

In 1993, he acted as the Administrative Assistant for the NATO Advanced Study Institute on Statistics and Neural Networks. Presently, he is working at a large multinational corporation based in St. Paul, Minnesota, in the areas of neural networks and statistical data analysis.