Efficient Algorithms for Function Approximation with Piecewise Linear Sigmoidal Networks

Don R. Hush *Senior Member IEEE* and Bill Horne *Member IEEE*

Abstract— **This paper presents a computationally efficient algorithm for function approximation with piecewise linear sigmoidal nodes. A one hidden layer network is constructed one node at a time using the well–known method of fitting the residual. The task of fitting an individual node is accomplished using a new algorithm that searches for the best fit by solving a sequence of Quadratic Programming problems. This approach offers significant advantages over derivative–based search algorithms (e.g. backpropagation and its extensions). Unique characteristics of this algorithm include: finite step convergence, a simple stopping criterion, solutions that are independent of initial conditions, good scaling properties and a robust numerical implementation. Empirical results are included to illustrate these characteristics.**

Keywords— **function approximation, nonlinear regression, constructive learning, piecewise linear sigmoids, multilayer perceptrons**

I. Introduction

Multilayer perceptrons (MLPs) have become a popular tool for approximating nonlinear functions in higher dimensions. Although they are not the panacea for these types of problems, they are clearly recognized as a useful member of the "toolbox of methods" that one might employ. Other methods include interaction splines [1], [2], [3], additive models [4], projection pursuit [5], [6], MARS [7], the Π method [8], hinging hyperplanes [9] and CART [10]. None of these are likely to perform consistently better than the others across a wide range of problems. At the same time however, it is nontrivial to develop a method that is truly effective in higher dimensions, and MLPs have found a useful niche in this arena. Both theoretical and practical reasons for this will be explored in subsequent sections of this paper.

Historically, MLPs have been plagued by slow learning. The desire to overcome this sluggishness has resulted in considerable work on the development of faster learning algorithms for MLPs. Backpropagation is a first–order local descent technique, closely resembling the stochastic gradient method. A majority of the work on faster algorithms has been concentrated on the development of more advanced first and second–order local descent methods (see [11] for an overview). These methods use first and/or second derivatives to determine directions of search that descend the criterion function as quickly as possible. All of these derivative–based descent methods possess the following characteristics. First their convergence is asymptotic, which means that it is not possible to bound the number of steps in the algorithm. It also makes it difficult to choose a suitable stopping criterion (i.e. one that is problem independent) [12], [13]. Second, these methods typically come with several user specified parameters that must be "tuned to the problem" if learning is to proceed at a reasonable rate (e.g. learning rates, momentum parameter, batch size, etc..). Finally, the solutions obtained with these methods depend on the initial conditions (because of local minima). Thus, to improve the likelihood of finding a good solution we must train the network from several different starting points.

The learning algorithm developed in this paper is quite different from the traditional family of derivative–based descent methods. First, a *constructive* approach is used, which builds the network one node at a time. The advantages of a constructive approach include computational efficiency and the ease of determining a suitable network size. In fact, there is theoretical evidence to suggest that the learning problem may be intrinsically easier if we are allowed to add nodes and weights during the learning process [14]. Although constructive approaches are not guaranteed to produce networks of *absolute minimal size*, there is good reason to believe that they can produce representations which are efficient [15]. Constructive algorithms have been the mainstay in the statistical community for many years, and in recent years many of these methods have been integrated into the neural network community [16], [8], [9], [17], [18], [7], [19], [20], [21], [22], [23], [24], [25].

Second, we use *piecewise linear* sigmoidal nodes instead of continuously differentiable logistic nodes. This changes the nature of the learning problem entirely. It becomes a *combinatorial* problem in the sense that the number of feasible solutions that we must search through to find a solution is *finite*. This makes it possible to develop learning algorithms that converge in a finite number of steps. In fact we derive polynomial bounds on the number of steps required for the algorithms that we develop. These algorithms also turn out to be quite easy to use. They have a simple (automatic) stopping criterion, and very few user specified parameters. In addition, they can be made to produce good solutions that are independent of initial conditions.

The remainder of this paper is organized as follows. The material in section II provides an introduction to the particular constructive approach used in this paper. Sections III and IV develop new learning algorithms for piecewise linear sigmoidal nodes, which constitute the primary contribution of this paper. Finally, empirical results are provided in section V to highlight some of the salient features of these new algorithms.

Don Hush is with the Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131

Bill Horne is with MakeWaves, Inc., 832 Valley Road, Watchung, NJ 07060

II. Function Approximation with Sigmoidal Basis Functions

The following notation is used in the development in this section. The symbol $\tilde{\mathbf{x}}$ is used to represent input vectors $\mathbf{x}$ that have been augmented with a 1 in the first position, that is

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \tag{1}$$

Similarly, $\tilde{\mathbf{w}}$ is used to represent weight vectors $\mathbf{w}_d$ that have been augmented with a "bias" weight in the first position

$$\tilde{\mathbf{w}} = \begin{bmatrix} w_0 \\ \mathbf{w}_d \end{bmatrix} \tag{2}$$

The dimension of these augmented vectors is $d+1$.

The class of function approximation problems addressed in this paper can be described as follows. The function $f : \mathbf{x} \to y$ defines a continuous nonlinear mapping from $\mathbf{x} \in \Re^d$ to $y \in \Re$ whose concise mathematical description is assumed unknown. Specific information about f is provided by way of a sample set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, which contains samples of f at a finite number of points in $\Re^d$. Using these samples (and our knowledge of f's general properties, e.g. continuity), our goal is to produce an estimate, $\hat{f}$, that approximates f as closely as possible.

The class of approximating functions considered here are one hidden layer neural networks of the form

$$\hat{f}_n(\mathbf{x}) = b_0 + \sum_{i=1}^{n} b_i g_i(\mathbf{x}) \tag{3}$$

where the $g_i(\mathbf{x})$ are sigmoidal functions. We work with different forms of the sigmoid in this paper, all of which are parameterized by a weight vector $\mathbf{w}_i$. This is made explicit with the notation $g_i(\mathbf{x}) = \sigma(\mathbf{x}, \mathbf{w}_i)$. The two most common realizations are the logistic function,

$$\sigma_l(\mathbf{x}, \tilde{\mathbf{w}}) = (1 + e^{-\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}})^{-1} \tag{4}$$

parameterized by the weight vector $\mathbf{w} = \tilde{\mathbf{w}}$ and the ramp function,

$$\sigma_r(\mathbf{x}, \mathbf{w}) = \begin{cases} 0 & \mathbf{w}_d^T \mathbf{x} \le \alpha \\ (\mathbf{w}_d^T \mathbf{x} - \alpha)/(\beta - \alpha) & \alpha \le \mathbf{w}_d^T \mathbf{x} \le \beta \\ 1 & \mathbf{w}_d^T \mathbf{x} \ge \beta \end{cases} \tag{5}$$

parameterized by $\mathbf{w}^T = [\alpha, \beta, \mathbf{w}_d^T]$. The ramp function can be interpreted as a piecewise linear approximation to the logistic function.

The models in (3), with sigmoidal basis functions, have been shown to be universal approximators over a compact subset S of $\Re^d$ for the class of *continuous* functions f from S to $\Re$ [26], [27]. These models have the familiar "linear combination of basis functions" form. It is well–known that when the basis functions are fixed, this form suffers from the curse of dimensionality. For example a typical bound on approximation error for fixed basis functions is $O(1/n^{1/d})$ [28], [29], indicating that it may require a number of basis functions that is exponential in d to achieve a significant reduction in error. When the basis functions are "tunable", however, it is possible (in principle) to circumvent the curse of dimensionality (or at least this aspect of it). For example, with a sigmoidal basis it has been shown that under very general conditions, the approximation error is bounded by $O(1/n)$ [28]. The missing $1/d$ in the power of the denominator is a strong motivation for the use of these basis functions in higher dimensional problems. In addition, Jones has shown that this $O(1/n)$ bound can be achieved *constructively* [15]. This result is also presented in [28], [30], where it receives a slightly different treatment. The proof of this result is itself constructive, and thus provides a framework for the development of an algorithm which can (in principle) achieve this bound.

> **Initialization:**
> $f_0(\mathbf{x}) = 0$
> **for** $n = 1$ **to** n_{max} **do**
> 1. Compute Residual: $e_n(\mathbf{x}) = f(\mathbf{x}) - f_{n-1}(\mathbf{x})$
> 2. Fit Residual: $g_n(\mathbf{x}) = \arg\min_{g \in G} \|e_n(\mathbf{x}) - g(\mathbf{x})\|$
> 3. Update Estimate: $f_n(\mathbf{x}) = \alpha f_{n-1}(\mathbf{x}) + \beta g_n(\mathbf{x})$
>   where α and β are chosen to minimize $\|f(\mathbf{x}) - f_n(\mathbf{x})\|$
>
> **endloop**

Fig. 1. Iterative Approximation Algorithm (IIA).

One manifestation of this algorithm is shown in Figure 1. It starts by fitting the first basis to the original function (the first time through the loop $e_1 = f$ and $f_1 = g_1$). The second basis is then fit to the residual from the first approximation, and the two are combined (in Step 3) to form the second approximation. This process of fitting a basis to the current residual and then combining it with the previous approximation continues until a suitable size model is found. This is called the *iterative approximation algorithm* because it builds the approximation by iterating on the residual (i.e. the unexplained portion of the function) at each step.

This algorithm is attractive in that the main loop contains only three steps, two of which are quite simple. The middle step however, can be quite difficult. This step requires that we find the function g_n that best fits the current residual. This problem generally does *not* have a closed form solution, and even algorithmic solutions are not guaranteed to produce the optimal g_n in an efficient manner. In this paper we develop algorithms for performing this step that produce "good approximations" (i.e. near optimal) in a computationally efficient manner.

Note that when sigmoidal functions are used in Step 2 of the IIA algorithm, they must be scaled and shifted so that they can better fit functions with arbitrary range and position. That is, the sigmoidal basis used in Step 2 of the IIA takes on the form

$$g(\mathbf{x}) = a_0 + a_1 \sigma(\mathbf{x}, \mathbf{w}) \tag{6}$$

where a_1 and a_0 are scaling and shifting parameters. The final model produced by the IIA algorithm can still be expressed in the form of equation (3), where the the b_i coefficients are simple deterministic functions of α,β, a_0 and

for $i = 1$ **to** n **do**
1. Compute f_{n-1}: $f_{n-1}(\mathbf{x}) = \sum_{j \neq i}^{n} a_j g_j(\mathbf{x})$
2. Compute Residual: $e_{n-1}(\mathbf{x}) = f(\mathbf{x}) - f_{n-1}(\mathbf{x})$
3. Fit Residual: $g_i(\mathbf{x}) = \arg\min_{g \in G} \|e_{n-1}(\mathbf{x}) - g(\mathbf{x})\|$
4. Update Estimate: $f_n(\mathbf{x}) = \sum_{j=1}^{n} a_j g_j(\mathbf{x})$
   where $\{a_i\}$ are chosen to minimize $\|f(\mathbf{x}) - f_n(\mathbf{x})\|$

endloop

Fig. 2. Refitting Algorithm.

a_1.

In practice it is common to use a *refitting* procedure to "fine tune" the result of IIA. This procedure can compensate somewhat for the suboptimal result that may be produced at Step 2, and also to some degree for limitations due to the constructive nature of IIA. A typical refitting procedure is shown in Figure 2. The basic idea is to refit each basis function, one at a time, to the residual formed from the approximation using the other $n - 1$ basis functions. This algorithm has the same attributes as the IIA: it optimizes individual basis functions by fitting them to a residual, and then reintegrates them into the overall fit. It differs from the IIA in that the residual is computed differently, and that the starting point for each refitting is usually close to its final point. This means that the search in Step 3 is generally very fast compared to its counterpart in Step 2 of the IIA. Because of this, refitting usually runs much faster than IIA.

III. Learning with Ramps and Hinging Sigmoids

This section develops learning algorithms for piecewise linear sigmoidal nodes. These algorithms can be used to optimize the basis in Step 2 of the IIA (and Step 3 of the refitting algorithm). We begin by defining a revised version of the ramp function called the *hinging sigmoid* (HS) function on which our algorithms are based. A HS node performs the function

$$\sigma_h(\mathbf{x}, \mathbf{w}) = \begin{cases} w_+, & \tilde{\mathbf{w}}_l^T \tilde{\mathbf{x}} \geq w_+ \\ \tilde{\mathbf{w}}_l^T \tilde{\mathbf{x}}, & w_- \leq \tilde{\mathbf{w}}_l^T \tilde{\mathbf{x}} \leq w_+ \\ w_-, & \tilde{\mathbf{w}}_l^T \tilde{\mathbf{x}} \leq w_- \end{cases} \tag{7}$$

where

$$\mathbf{w} = \begin{pmatrix} \tilde{\mathbf{w}}_l \\ w_+ \\ w_- \end{pmatrix} \tag{8}$$

This function is identical to the ramp basis defined in (5) and (6), but is parameterized differently. An example of the surface formed by an HS node on a two–dimensional input is shown in Figure 3. It is comprised of three hyperplanes joined pairwise continuously at two hinge locations. The upper and middle hyperplanes are joined at "Hinge 1" and the lower and middle hyperplanes are joined at "Hinge 2". These hinges induce linear partitions on the input space that divide the space into three regions, and the samples

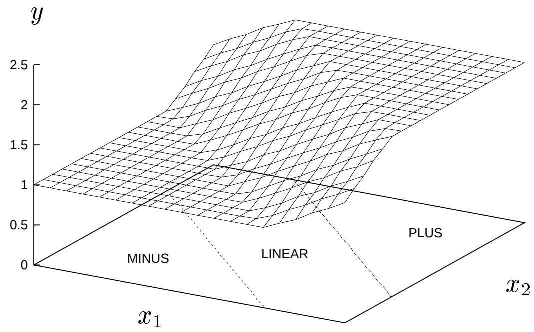

Fig. 3. A Sigmoid Hinge function in two dimensions.

in S into three subsets,

$$\begin{aligned} S_+ &= \{(\mathbf{x}_i, y_i) : \tilde{\mathbf{w}}_l^T \tilde{\mathbf{x}}_i \geq w_+\} \\ S_l &= \{(\mathbf{x}_i, y_i) : w_- \leq \tilde{\mathbf{w}}_l^T \tilde{\mathbf{x}}_i \leq w_+\} \\ S_- &= \{(\mathbf{x}_i, y_i) : \tilde{\mathbf{w}}_l^T \tilde{\mathbf{x}}_i \leq w_-\} \end{aligned} \tag{9}$$

These subsets, and the corresponding regions of the input space, are referred to as the PLUS, LINEAR and MINUS subsets/regions respectively. We refer to this type of partition as a *sigmoidal partition.* A sigmoidal partition of S will be denoted $P = \{S_+, S_l, S_-\}$, and the set of all such partitions will be denoted $\Pi = \{P_i\}$.

Input samples which fall on the boundary between two regions can be assigned to the set on either side. These points are referred to as *hinge samples* and play a crucial role in subsequent development.

Note that once a weight vector $\mathbf{w}$ in (8) is specified, the partition P is completely determined, but the reverse is not necessarily true. That is, there are generally an infinite number of weight vectors that induce the same partition.

We begin our quest for a learning algorithm with the development of an expression for the empirical risk. The empirical risk (squared error over the sample set) is defined

$$E_P(\mathbf{w}) = \frac{1}{2} \sum_S \left(y_i - \sigma_h(\mathbf{x}_i, \mathbf{w})\right)^2 \tag{10}$$

This expression can be expanded into three terms, one for each set in the partition,

$$E_P(\mathbf{w}) = \frac{1}{2} \sum_{S_-} (y_i - w_-)^2 + \frac{1}{2} \sum_{S_+} (y_i - w_+)^2 + \frac{1}{2} \sum_{S_l} (y_i - \tilde{\mathbf{w}}_l^T \tilde{\mathbf{x}}_i)^2$$

After further expansion and rearrangement of terms we obtain

$$\begin{aligned} E_P(\mathbf{w}) = \; & \tfrac{1}{2}\left(N_- w_-^2 + N_+ w_+^2 + \tilde{\mathbf{w}}_l^T \mathbf{R}_l \tilde{\mathbf{w}}_l\right) \\ & - w_- s_y^- - w_+ s_y^+ - \tilde{\mathbf{w}}_l^T \mathbf{r}_l + s_y^2 \end{aligned}$$

where

$$\mathbf{R}_l = \textstyle\sum_{S_l} \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^T \quad \mathbf{r}_l = \sum_{S_l} \tilde{\mathbf{x}}_i y_i \tag{11}$$

$$s_y^2 = \frac{1}{2}\sum_S y_i^2 \quad s_y^+ = \sum_{S_+} y_i \quad s_y^- = \sum_{S_-} y_i \tag{12}$$

and N_+, N_l and N_- are the number of samples in S_+, S_l and S_- respectively. This expression can be simplified to

$$E_P(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{R}\mathbf{w} - \mathbf{w}^T\mathbf{r} + s_y^2 \tag{13}$$

where

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}_l & 0 & 0 \\ 0 & N_+ & 0 \\ 0 & 0 & N_- \end{pmatrix} \quad \mathbf{r} = \begin{pmatrix} \mathbf{r}_l \\ s_y^+ \\ s_y^- \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} \tilde{\mathbf{w}}_l \\ w_+ \\ w_- \end{pmatrix} \tag{14}$$

The subscript P is used to emphasize that this criterion is dependent on the partition (i.e. P is required to form $\mathbf{R}$ and $\mathbf{r}$). In fact, the nature of the partition plays a critical role in determining the properties of the solution. More specifically it determines the rank of $\mathbf{R}$. Note that $\mathbf{R}$ is a symmetric matrix and in general is positive *semidefinite*. When $\mathbf{R}$ is positive *definite* (i.e. full rank), P is referred to as a *stable* partition, and when $\mathbf{R}$ has reduced rank P is referred to as an *unstable* partition. A stable partition requires that there be at least one sample in S_+ and S_-, and that the samples in S_l form a positive definite correlation matrix $\mathbf{R}_l$. The conditions on S_+ and S_- are quite reasonable, given that an empty set in either case would mean that the corresponding region of the sigmoid could take on an arbitrary value without affecting the empirical error. A necessary (but not sufficient) condition for $\mathbf{R}_l > 0$ is that there be at least $d+1$ samples in S_l. [1] For purposes of algorithm development we require that $|S_l| > N_{min}$, where N_{min} is a suitably chosen value greater than or equal to $d+1$. With the proper choice of N_{min} we can often insure that $\mathbf{R}_l$ is not only positive definite, but also well–behaved.

Alternatively, we could consider adding a regularization term to $E_P(\mathbf{w})$ of the form $\lambda\|\mathbf{w}\|^2$ (e.g. weight decay). The corresponding empirical risk would have the same form as (13) with $\mathbf{R}$ replaced by $\mathbf{R}_\lambda = \mathbf{R} + \lambda\mathbf{I}$. Note that choosing $\lambda > 0$ guarantees $\mathbf{R}_\lambda > 0$. In this case *all* partitions are stable and there is no need to monitor the number of samples in S_-, S_l and S_+. On the other hand, adding the regularization term leads to a *biased* solution. The bias can be minimized however, by making λ sufficiently small. [2]

In summary, when seeking a minimizing solution for $E_P(\mathbf{w})$ we restrict ourselves to stable partitions because of the potential nonuniqueness associated with solutions to unstable partitions. If we use a regularization term (i.e. $\lambda > 0$), then stable partitions are guaranteed. On the other hand, in practice we can often circumvent the need for regularization by simply requiring that $|S_+| \geq 1$, $|S_-| \geq 1$ and $|S_l| \geq N_{min}$.

Note that $E_P(\mathbf{w})$ is quadratic in $\mathbf{w}$, and with $\mathbf{R}_\lambda > 0$ a unique global minimum is guaranteed. Thus, it would seem that the value of $\mathbf{w}$ that minimizes $E_P(\mathbf{w})$ could be readily obtained by solving the system of linear equations $\mathbf{R}_\lambda\mathbf{w} = \mathbf{r}$. However, the solution to $\mathbf{R}_\lambda\mathbf{w} = \mathbf{r}$ does not necessarily minimize $E_P(\mathbf{w})$ because the resulting $\mathbf{w}$ may induce a different partition on S which changes the expression for $E_P(\mathbf{w})$. Determining a weight vector that simultaneously minimizes $E_P(\mathbf{w})$ and preserves the current partition can be posed as a constrained optimization problem. This problem takes on the form

$$\begin{array}{l} \min \frac{1}{2}\mathbf{w}^T\mathbf{R}_\lambda\mathbf{w} - \mathbf{w}^T\mathbf{r} \\ \text{subject to } \mathbf{A}\mathbf{w} \leq \mathbf{0} \end{array} \tag{15}$$

where the inequality constraints are designed to maintain the current partition. Using the partition equations in (9) we obtain the following form for $\mathbf{A}$.

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_+ \\ \mathbf{A}_{l+} \\ \mathbf{A}_{l-} \\ \mathbf{A}_- \end{pmatrix} \tag{16}$$

where the rows of $\mathbf{A}_+$,$\mathbf{A}_{l+}$,$\mathbf{A}_{l-}$ and $\mathbf{A}_-$ are

$$\begin{array}{llll} \mathbf{a}_+^T & = (\; -\tilde{\mathbf{x}}_i^T \quad 1 \quad 0 \;), & \tilde{\mathbf{x}}_i \in S_+ \\ \mathbf{a}_{l+}^T & = (\; \tilde{\mathbf{x}}_i^T \quad -1 \quad 0 \;), & \tilde{\mathbf{x}}_i \in S_l \\ \mathbf{a}_{l-}^T & = (\; -\tilde{\mathbf{x}}_i^T \quad 0 \quad 1 \;), & \tilde{\mathbf{x}}_i \in S_l \\ \mathbf{a}_-^T & = (\; \tilde{\mathbf{x}}_i^T \quad 0 \quad -1 \;), & \tilde{\mathbf{x}}_i \in S_- \end{array} \tag{17}$$

Note that there are two constraints associated with each sample in S_l so that $\mathbf{A}$ has a total of $N_+ + 2N_l + N_-$ rows. The optimization problem in (15) is a *Quadratic Programming* problem with *inequality constraints*, and because $\mathbf{R}_\lambda > 0$ it has a unique global minimum. The general Quadratic Programming problem is NP–hard [31] and also hard to approximate [32]. However, the convex case which we restrict ourselves to here (i.e. $\mathbf{R}_\lambda > 0$) admits a polynomial time solution [33]. In this paper we use the *active set* algorithm described in [13] to solve (15). This algorithm is similar to the Simplex algorithm for Linear Programming in that it is simple, robust, and guaranteed to converge in a finite number of steps. It also tends to run very efficiently in practice. With the proper implementation, this algorithm runs in $O(k(d^2 + Nd))$ time, where k is the number of times through the main loop. Although k can grow quite large in theory, in practice it is typically on the order of d or less.

The solution to the quadratic programming problem in (15) is only as good as the current partition allows. The more challenging aspect of minimizing $E_P(\mathbf{w})$ is in the search for a good partition. Unfortunately there is no simple arrangement of partitions that corresponds to a partial ordering in $E_P(\mathbf{w})$, so the search for the optimal partition will be a computationally challenging problem. An exhaustive search is usually out of the question because of the prohibitively large number of partitions, as given by the following lemma.

Lemma 1: Let S contain a total of N samples in $\Re^d$ that lie in general position (i.e. no $d+1$ of the samples lie

[1] In truth, the positive definite condition on $\mathbf{R}_l$ is not critical to our implementation, and can be removed if we are willing to employ techniques for manipulating reduced rank matrices. In fact we are often forced to employ such techniques in practice to handle ill–conditioned matrices that arise.

[2] Alternatively, in practice it is customary to choose λ to optimize the bias/variance trade–off.

on a $d-1$ dimensional hyperplane). Then the number of sigmoidal partitions defined in (9) is $\Theta(N^{d+1})$.

Proof: Let $L(N,d)$ represent the number of linear dichotomies of N points in d dimensions. When the N points are in general position it is well–known that $L(N,d) = \Theta(N^d)$ [34]. Each sigmoidal partition is comprised of two linear dichotomies, one formed by Hinge 1 and the other by Hinge 2, and these dichotomies are constrained to be simple translations of one another. That is, the separating hyperplanes that induce these two dichotomies are constrained to be parallel. Consequently, when one is given, the other can take on at most $N+1$ distinct values. These come from the $N+1$ dichotomies induced as a hyperplane with fixed orientation is swept from one end of the input data to the other (a new partition is induced each time it crosses a data sample). Thus, there are at most $O(N^{d+1})$ sigmoidal partitions. Now we show that there are at least this many. Consider an arbitrary dichotomy formed by Hinge 1. It splits S into two sets of size N_1 and N_2. Suppose the first set is S_+ and the second is $S_l \cup S_-$. Because the LINEAR region must fall between the PLUS and MINUS regions, the "legal" dichotomies for Hinge 2 can only be obtained by sweeping through the samples in the second set. This gives N_2+1 dichotomies (including those leading to unstable partitions). Similarly, if the second set is labeled S_+ and the first is $S_l \cup S_-$ then there will be N_1+1 possible dichotomies for Hinge 2. These two cases have one dichotomy in common, so in total, for each pair of dichotomies induced by Hinge 1 there will be $N_1+N_2+1 = N+1$ different dichotomies for Hinge 2. Thus, there are $(N+1)L(N,d)/2 = \Omega(N^{d+1})$ total sigmoidal partitions. ∎

Even though the number of sigmoidal partitions is polynomial in N, exhaustive search is clearly out of the question for even modest values of N and d. Lacking a global search strategy which is provably more efficient we are forced to consider heuristic methods for searching Π. Two such algorithms are described below. The first was introduced elsewhere and is called the `Ramps` algorithm. `Ramps` searches through partitions using a strategy analogous that used by the K–Means clustering algorithm. The second is a new method introduced here which searches through partitions by allowing a small number of carefully chosen points to migrate from one set to another at each step. These points are chosen so that the fit can be improved on each successive partition.

The `Ramps` algorithm presented here is due to Friedman and Breiman [35]. The advantage of this algorithm lies in its simplicity. It searches through partitions by repeatedly solving $\mathbf{Rw} = \mathbf{r}$ until it converges to a point where $\mathbf{w}$ induces the same partition on two consecutive iterations (note that $\lambda = 0$ for this algorithm). The complete algorithm, as presented in [35], is shown in Figure 4, where the ramp basis notation in (5) and (6) has been used. This algorithm has the advantage that, when it converges, convergence is usually very fast. Unfortunately it does not always converge. Note that there is no mechanism to prevent the algorithm from diverging to an unstable partition, and although there are relatively few such partitions, divergence to one of them is commonly observed in practice. This behavior can be illustrated with a simple one–dimensional example. Consider the five sample problem in Figure 5. There are three stable partitions for this problem,

{*Invoke routine with feasible solution* $W = \{\mathbf{w}_d, \alpha, \beta, a_0, a_1\}$.}

procedure `Ramps` (W)
 repeat
  Compute z *and Partition data into* S_-, S_l *and* S_+.
   $z_i = \mathbf{w}_d^T \mathbf{x}_i,\ i = 1,2,...,N.$
   $S_- = \{(\mathbf{x}_i, y_i) : z_i < \alpha\}$
   $S_l = \{(\mathbf{x}_i, y_i) : \alpha \le z_i \le \beta\}$
   $S_+ = \{(\mathbf{x}_i, y_i) : \beta < z_i\}$
  Compute $\mathbf{R}$, $\mathbf{r}$, β_- *and* β_+.
   $\mathbf{R} = (\sum_{S_l} \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^T)/N_l$
   $\mathbf{r} = (\sum_{S_l} \tilde{\mathbf{x}}_i y_i)/N_l$
   $\beta_- = (\sum_{S_-} y_i)/N_-$
   $\beta_+ = (\sum_{S_+} y_i)/N_+$
  Update $\tilde{\mathbf{w}}^T = (w_0\ \mathbf{w}_d^T)$.
   $\tilde{\mathbf{w}} = \mathbf{R}^{-1}\mathbf{r}$
  Update α *and* β.
   $\alpha = (\beta_- - w_0)/\|\mathbf{w}_d\|$
   $\beta = (\beta_+ - w_0)/\|\mathbf{w}_d\|$
  Normalize $\mathbf{w}_d$.
   $\mathbf{w}_d = \mathbf{w}_d/\|\mathbf{w}_d\|$
 until ($\mathbf{w}_d$,α,β converge) ;
 Compute the bias and scale parameters.
  $a_0 = \beta_-$
  $a_1 = \beta_+ - \beta_-$

 return(W);
end ; {`Ramps`}

Fig. 4. The `Ramps` Algorithm.

$$P_1 = \{\{(1,1)\}, \{(2,4),(3,4.5),(4,5)\}, \{(5,8)\}\}$$

$$P_2 = \{\{(1,1),(2,4)\}, \{(3,4.5),(4,5)\}, \{(5,8)\}\}$$

$$P_3 = \{\{(1,1)\}, \{(2,4),(3,4.5)\}, \{(4,5),(5,8)\}\}$$

Starting from any one of these three partitions, the `Ramps` algorithm will diverge to an unstable partition (with zero points in S_+ and S_-) after just one step. In practice the simplest way to compensate for this behavior is to restart the algorithm from a different initial partition, and repeat this process if necessary until convergence to a stable partition is obtained. The `Ramps` algorithm can also be shown to exhibit limit cycles (with even periods). In practice we can compensate for this behavior by placing an upper limit on the total number of iterations.

The partitions produced by `Ramps` can vary dramatically from one iteration to the next (i.e. a large number of samples can move between subsets). The algorithm considered next is a more conservative, in that only a few carefully chosen samples are allowed to move between subsets. This approach employs a Quadratic Programming (QP) algorithm at each new partition to determine the optimal weight vector for that partition (i.e. the optimal orientation for the separating hyperplanes). Transitions are made from one partition to the next by allowing hinge samples to flip from

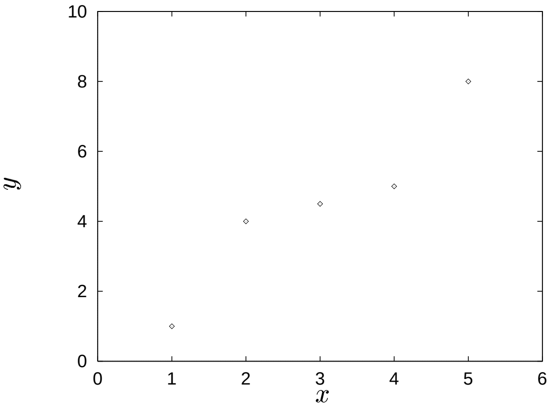

Fig. 5. Example problem for the `Ramps` algorithm.

{*Invoke with feasible solution* $W = \{\mathbf{w}, \mathbf{R}_\lambda, \mathbf{r}, A, S_+, S_l, S_-\}$.}

procedure `HingeDescent` (W)
    {*Walk hinges across data until a min partition is found.*}
    $E = \frac{1}{2}\mathbf{w}^T\mathbf{R}_\lambda\mathbf{w} - \mathbf{w}^T\mathbf{r}$
    do
        $E_{min} = E$

        {*Flip Hinge 1 Samples.*}
        for each $((\mathbf{x}_i, y_i)$ on Hinge 1$)$ **do**
            if $((\mathbf{x}_i, y_i) \in S_+$ **and** $N_+ > 1)$ **then**
                Move $(\mathbf{x}_i, y_i)$ from S_+ to S_l
                Update $\mathbf{R}_\lambda$, $\mathbf{r}$, and A
            elseif $((\mathbf{x}_i, y_i) \in S_l$ **and** $N_l > N_{min})$ **then**
                Move $(\mathbf{x}_i, y_i)$ from S_l to S_+
                Update $\mathbf{R}_\lambda$, $\mathbf{r}$, and A
            endif
        endloop

        {*Flip Hinge 2 Samples.*}
        for each $((\mathbf{x}_i, y_i)$ on Hinge 2$)$ **do**
            if $((\mathbf{x}_i, y_i) \in S_-$ **and** $N_- > 1)$ **then**
                Move $(\mathbf{x}_i, y_i)$ from S_- to S_l
                Update $\mathbf{R}_\lambda$, $\mathbf{r}$, and A
            elseif $((\mathbf{x}_i, y_i) \in S_l$ **and** $N_l > N_{min})$ **then**
                Move $(\mathbf{x}_i, y_i)$ from S_l to S_-
                Update $\mathbf{R}_\lambda$, $\mathbf{r}$, and A
            endif
        endloop

        {*Compute optimal solution for new partition.*}
        $W =$ `QPSolve`(W);

        $E = \frac{1}{2}\mathbf{w}^T\mathbf{R}_\lambda\mathbf{w} - \mathbf{w}^T\mathbf{r}$

    while $(E < E_{min})$;

    return(W);
end ; {`HingeDescent`}

Fig. 6. The `HingeDescent` Algorithm. When $\lambda > 0$ the tests for $N_- > 1$, $N_+ > 1$ and $N_l > N_{min}$ can be omitted.

one side of the hinge boundary to the next. The search is terminated when a minimum value of $E_P(\mathbf{w})$ is found (i.e. it can no longer be reduced by flipping hinge samples). The motivation for this algorithm originates from the following observations regarding the types of solutions produced by the QP algorithm. The QP solution for an individual partition can take on two different forms.

1. In the first form the solution has no active constraints, that is none of the rows of $\mathbf{A}$ satisfy $\mathbf{a}_i^T\mathbf{w} = 0$. This means that there are no hinge samples associated with this solution. Consequently this solution is a local minimum over Π. This is easily verified, since a small perturbation of $\mathbf{w}$ towards any another partition will increase $E_P(\mathbf{w})$.

2. In the second form the QP solution has one or more active constraints, i.e. one or more hinge samples. In this case it may be possible to reduce $E_P(\mathbf{w})$ further by perturbing $\mathbf{w}$ in a direction that violates one of the active constraints. To make such a perturbation legal we can simply flip the sign of the constraint (multiply by -1). Note that it is always possible to flip the sign of an active constraint and maintain feasibility of the current solution (if $\mathbf{a}^T\mathbf{w} = 0$ then $-\mathbf{a}^T\mathbf{w} = 0$). Flipping the sign of a constraint is equivalent to moving a hinge sample across the hinge boundary from one set to another (e.g. from S_l to S_+). This results in a new partition of the data. Computationally this involves a relabeling of the data sample, flipping the sign of the constraint, and updating $\mathbf{R}_\lambda$ and $\mathbf{r}$. After the flip, a new QP solution is sought. If this new solution is different from the previous solution then it will have a reduced value of $E_P(\mathbf{w})$. On the other hand, if it is the same as the previous solution then it represents a local minimum over Π. We know this to be true because it is the minimum for two adjacent partitions which implies that a perturbation in *any* direction will increase $E_P(\mathbf{w})$.

It is relatively straightforward to synthesize an algorithm for descending $E_P(\mathbf{w})$ that capitalizes on these properties of the QP solution. For example, the algorithm in Figure 6 moves from one partition to the next (and from one QP solution to the next) by successively flipping hinge samples across the hinge boundaries as described above until it reaches a local minimum over Π. We call this the `HingeDescent` algorithm because it allows the hinges to "walk across" the data in a manner that descends the $E_P(\mathbf{w})$ criterion. Note that provisions are made within the algorithm to avoid unstable partitions in the event that $\lambda = 0$. Note also that it is easy to modify this algorithm to descend only one hinge at a time, simply by omitting one of the blocks of code that flips samples across the corresponding hinge boundary.

Lemma 2: With $\lambda > 0$ the `HingeDescent` algorithm will converge to a stable partition of $E_P(\mathbf{w})$ in a finite number of steps.

Proof: First note that $\lambda > 0$ guarantees $\mathbf{R}_\lambda > 0$, so that a QP solution for *any* partition can be found in a finite number of steps. For example, it is relatively easy to show that the active set algorithm satisfies this condition. The proof of this result is beyond the scope of this paper, but can easily be found in the literature [12], [13]. Now, by design, `HingeDescent` always moves from one partition to the next, reducing $E_P(\mathbf{w})$ at each step (except the last one) so that no partitions are revisited. Since there are a finite number of partitions (see Lemma 1) this algorithm must terminate in a finite number of steps. QED. ■

Assume that `QPSolve` runs in $O(k(d^2+Nd))$ time as previously stated. Then the run time of `HingeDescent` is given by $O(N_p((k+N_h)d^2 + kNd))$, where N_h is the number of

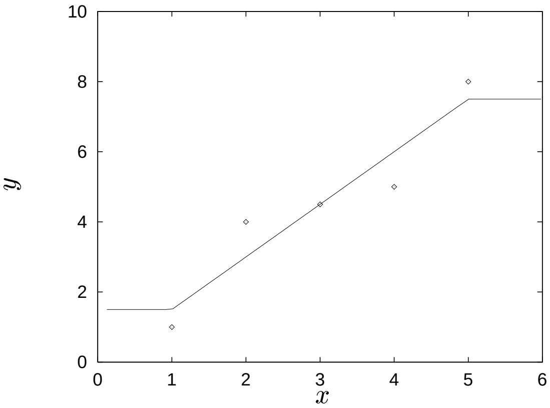

Fig. 7. Solution to the 5 sample problem using `HingeDescent`.

samples flipped at each step and N_p is the total number of partitions explored. Typical values for k and N_h are on the order of d, simplifying this expression to $O(N_p(d^3 + Nd^2))$. N_p can vary widely, but is often substantially less than N. In contrast the `Ramps` algorithm, if properly implemented, runs in $O(N_p N_h d^2)$, where N_h is the number of samples that move between sets on each iteration and N_p is the total number of partitions explored. For `Ramps`, N_h is often much larger than d, so its run time is roughly comparable to that of `HingeDescent`.

In contrast however, `HingeDescent` is capable of providing solutions which are not possible with `Ramps`. For example, consider the problem illustrated earlier in Figure 5. `Ramps` is not able to provide a stable solution for this problem, but `HingeDescent` finds the optimal solution (shown in Figure 7) from any valid starting point.

Both `Ramps` and `HingeDescent` seek a local minimum over Π, and both can produce poor solutions depending on their starting partition. One way to remedy this is to start them from several different initial partitions, and then retain the best solution overall. We take a different approach in the next section where we present an algorithm that always starts with the same initial condition, visits several local minima along the way, and always ends up with the same final solution each time. The `SweepingHinge` algorithm as it is called, builds on the approach adopted in the `HingeDescent` algorithm.

{*Returns initial feasible solution* $W = \{\mathbf{w}, \mathbf{R}, \mathbf{r}, A, S_+, S_l, S_-\}$.}

procedure `InitialLinearFit` (S)
{*Compute Least Squares Fit to data.*}
$\mathbf{R}_l = \sum_{i=1}^{N} \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^T$
$\mathbf{r}_l = \sum_{i=1}^{N} \tilde{\mathbf{x}}_i y_i$
$\tilde{\mathbf{w}}_l = \mathbf{R}_l^{-1} \mathbf{r}_l$

{*Find the two samples at the extremes.*}
for each $(\mathbf{x}_i \in S)$ **do**
$z_i = \tilde{\mathbf{w}}_l^T \tilde{\mathbf{x}}_i$
endloop
$j = \arg\min_i\{z_i\}$
$k = \arg\max_i\{z_i\}$

{*Position Hinge 1 on the max sample and put in* S_+. }
$S_+ = \{(\mathbf{x}_k, y_k)\}$
$N_+ = 1$
$A = \{(\ -\tilde{\mathbf{x}}_k^T \quad 1 \quad 0 \ )\}$
$w_+ = z_k$
$\mathbf{R}_l = \mathbf{R}_l - \tilde{\mathbf{x}}_k \tilde{\mathbf{x}}_k^T$
$\mathbf{r}_l = \mathbf{r}_l - \tilde{\mathbf{x}}_k y_k$
$s_y^+ = y_k$

{*Position Hinge 2 on the min sample and put in* S_-. }
$S_- = \{(\mathbf{x}_j, y_j)\}$
$N_- = 1$
$A = A \cup \{(\ \tilde{\mathbf{x}}_j^T \quad 0 \quad -1 \ )\}$
$w_- = z_j$
$\mathbf{R}_l = \mathbf{R}_l - \tilde{\mathbf{x}}_j \tilde{\mathbf{x}}_j^T$
$\mathbf{r}_l = \mathbf{r}_l - \tilde{\mathbf{x}}_j y_j$
$s_y^- = y_j$

return(W);
end ; {`InitialLinearFit`}

Fig. 8. The `InitialLinearFit` Algorithm.

IV. The Sweeping Hinge Algorithm

The `SweepingHinge` algorithm works as follows. It starts by performing a linear fit to the data. This fit is used for the initial linear region of the sigmoid. The initial hinges are then placed at the two extreme samples on opposite ends of the linear fit. This puts one sample in the PLUS region and one in the MINUS region, leaving $N-2$ samples in the LINEAR region. The details of `InitialLinearFit`, including the initialization of all relevant algorithm parameters, are shown in Figure 8. Note that this initialization is analogous to the use of small weights in the initialization of the backpropagation algorithm, since small weights tend to place data in the linear region of the sigmoid.

After the initial linear fit, the hinges are allowed to descend to a local minimum using `HingeDescent`. This corresponds loosely to the solution that would be produced by backpropagation. The `SweepingHinge` algorithm continues to look past this solution for a better one. This is accomplished by *sweeping* Hinge 1 across the data one sample at a time. Mechanically this is achieved by moving one additional sample from S_l to S_+ at each step. Hinge 2 is allowed to descend to an optimal position at each of these steps using the `Hinge2Descent` algorithm. This algorithm is identical to `HingeDescent` except that the code that flips samples across Hinge 1 is omitted. The best overall solution from the sweep is retained and "fine–tuned" with one final pass through the `HingeDescent` algorithm to produce the final solution. The complete algorithm is shown in Figure 9.

All of the partitions explored by the `SweepingHinge` algorithm are determined in a data driven fashion. At the same time, the mandatory sweep of Hinge 1 forces this algorithm to explore a rich set of partitions. In fact, this algorithm tends to pass through basins of attraction for several local minima during the sweep. In addition, the forced sweep guarantees that the algorithm will terminate in finite time. If the number of data samples is large, the sweep time can be reduced by moving M samples into S_+ at each step (instead of 1). In fact, when N is large compared to d, there is often little difference between solutions

```
{This routine returns a solution W* = {w, R, r, A, S+, Sl, S-}.}

procedure SweepingHinge (S)
    {Find an Initialize Feasible Solution and
    then search for a minimizing partition.}
    W = InitialLinearFit(S);
    W = HingeDescent(W);

    {Save current solution and compute its criterion.}
    W* = W
    E* = 1/2 w^T R w - w^T r

    {Sweep Hinge 1 Across the Data.}
    while (N+ < N - Nmin - 1) do

        {Find the sample in Sl closest to
        Hinge 1 and move it to S+.}
        for each ((xi, yi) ∈ Sl) do
            zi = w̃l^T x̃i
        endloop
        j = arg min_i{zi}
        Move (xj, yj) from Sl to S+, and update R,r,A.
        w+ = zj

        {Maintain a stable partition by moving
        a sample from S- to Sl if needed.}
        if (Nl < Nmin) then
            for each ((xi, yi) ∈ S-) do
                zi = w̃l^T x̃i
            endloop
            j = arg max_i{zi}
            Move (xj, yj) from S- to Sl, and update R,r,A.
            w- = zj
        endif

        {Optimize Hinge 2 location.}
        W = Hinge2Descent(W);

        {Keep track of the best solution so far.}
        E = 1/2 w^T R w - w^T r
        if (E < E*) then
            E* = E
            W* = W
        endif
    endloop

    {Fine-tune best solution from the sweep.}
    W* = HingeDescent(W*);
    return(W*);
end ;    {SweepingHinge}
```

Fig. 9. The `SweepingHinge` Algorithm.

for neighboring partitions and it is possible to sweep across several samples at each step without adversely affecting the final solution. Note that this algorithm always starts at the same position (determined by the data), and as long as M is fixed this algorithm always produces the same answer. While there is no guarantee that it will locate the global minimum over Π, there is good reason to believe that it will provide solutions of high quality. This claim is supported in part by the empirical results presented in section V.

The run time of `SweepingHinge` is no worse than N times that of `HingeDescent` (it is usually much less). Given this, the run time for this algorithm is $O(NN_p(d^3+Nd^2))$. Consequently, `SweepingHinge` scales reasonably well in both N and d, considering the nature of the problem it is designed to solve.

V. Empirical Results

Three sets of empirical results are presented in this section. The first involves a two–dimensional function whose approximations can be displayed visually and compared with results in [8], [9], [24]. The second is an experiment designed to test the effect of dimensionality on the models produced by the IIA/`SweepingHinge` algorithm. This experiment can be viewed as an empirical test of Barron's theoretical bound on approximation error. The third experiment involves a comparison with several other nonparametric modeling methods from [36].

All of the results presented in this section used the IIA algorithm to build a one–hidden layer network model. When `SweepingHinge` was employed at Step 2 of the algorithm, λ was set to zero and N_{min} was set equal to $3d$, where d is the input dimension. This proved to be sufficient to maintain both stable and well–behaved partitions. For comparison, `HingeDescent` and `Ramps` were also employed at Step 2. `HingeDescent` was performed from 10 random initial hinge locations in each case, and the best result incorporated into the model. Because of the convergence problems of `Ramps`, it was restarted from random hinge locations as often as necessary to produce 10 stable solutions in each case, and the best result incorporated into the model. Finally, a refitting pass (Figure 2) was employed after each new node was added in the IIA. The refitting algorithm used `HingeDescent` (or `Ramps`) to "re–fit" the residual at Step 3.

Results of the `SweepingHinge` algorithm applied to the two–dimensional function in Figure 10 are shown in Figures 11–13. A total of 200 randomly chosen samples from this function were used for training, and the fits were produced with 4, 8 and 12 nodes. These results are comparable to those obtained using other methods on this same (or similar) problem(s) (e.g. see the results in [8], [9], [24]). It is also instructive to examine the effects of noise. Figures 14–17 show results analogous to those in Figures 10–13, except that uniform noise with a variance of $\sigma_n^2 = 0.025$ has been added to the target function [3]. These results illustrate a remarkable ability of these models to extract the underlying deterministic function in the presence of noise.

To provide a quantitative comparison, `HingeDescent` and `Ramps` were also applied to this function. An independent test set of 1600 samples was used to measure the generalization performance. Results for $\sigma_n^2 = 0$, 0.025 and 0.1 are provided in Tables I, II and III. The results for `HingeDescent` and `Ramps` are averaged over 10 trials and the standard deviation is shown in parentheses. (`SweepingHinge` always provides the same result by design).

Note first that the performances of `SweepingHinge` and `HingeDescent` are consistently better than those of `Ramps`. One might expect `HingeDescent` and `Ramps` to produce similar results, since both descend to a local minimum from random starting points. The difference however is that `HingeDescent` is guaranteed to converge from any (stable)

[3] Note that the total energy in the noise–free target function is 0.5.

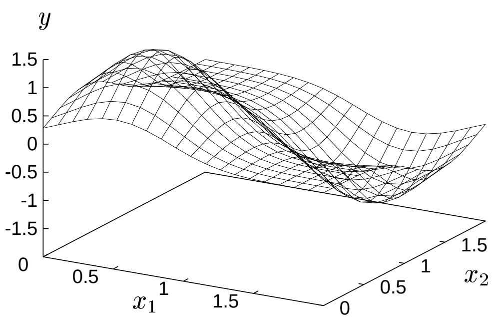

Fig. 10. Original Function.

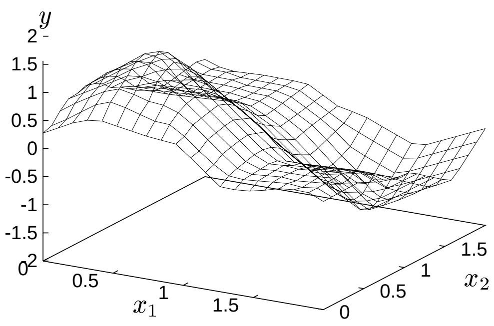

Fig. 13. Fit with 12 nodes.

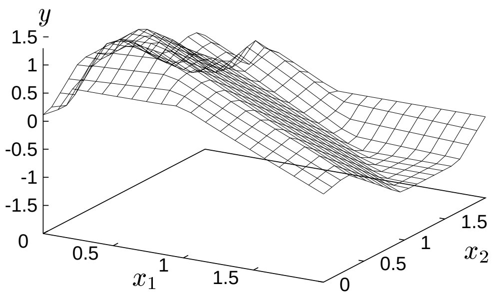

Fig. 11. Fit with 4 Nodes.

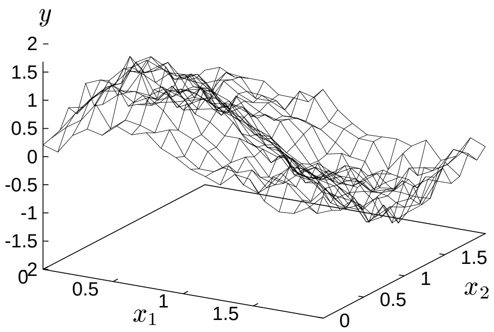

Fig. 14. Original function with noise.

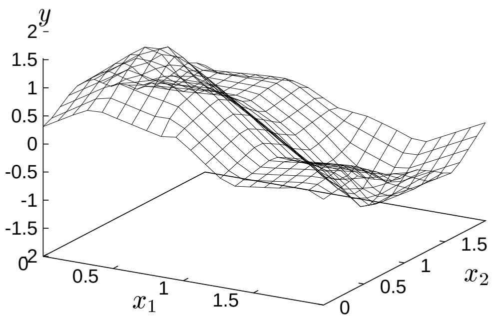

Fig. 12. Fit with 8 nodes.

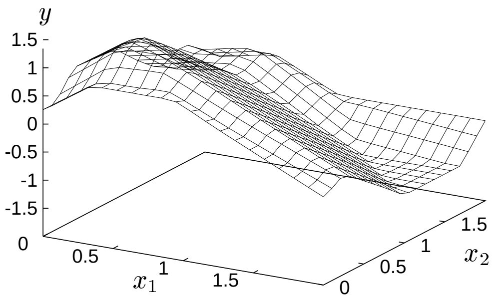

Fig. 15. Fit to noisy function with 4 nodes.

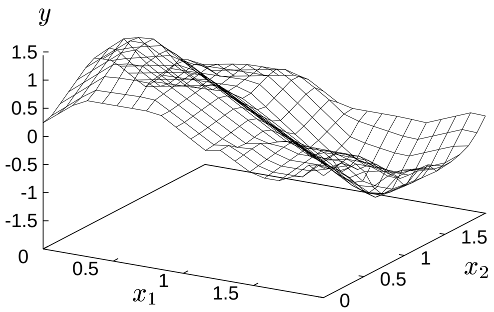

Fig. 16. Fit to noisy function with 8 nodes.

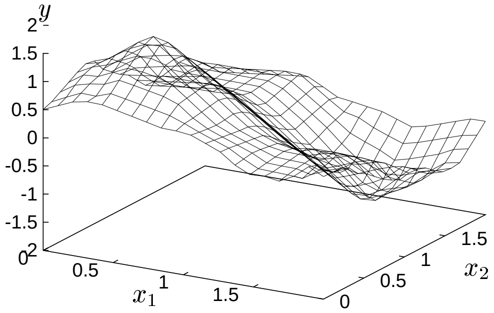

Fig. 17. Fit to noisy function with 12 nodes.

Data Set	Number of Nodes	MSE for Sweeping Hinge	Avg (Std) MSE for Hinge Descent	Avg (Std) MSE for RAMPS
Train	4	0.034	0.029 (0.0070)	0.038 (0.0041)
	8	0.0045	0.0013 (0.00023)	0.011 (0.0077)
	12	0.00086	0.00074 (0.00013)	0.0047 (0.0030)
Test	4	0.045	0.041 (0.011)	0.050 (0.0055)
	8	0.0060	0.0024 (0.00039)	0.016 (0.011)
	12	0.0019	0.0017 (0.00026)	0.0081 (0.0058)

TABLE I
Mean–Squared Error (MSE) comparisons with $\sigma_n^2 = 0$.

Data Set	Number of Nodes	MSE for Sweeping Hinge	Avg (Std) MSE for Hinge Descent	Avg (Std) MSE for RAMPS
Train	4	0.093	0.049 (0.0067)	0.066 (0.010)
	8	0.024	0.024 (0.0017)	0.041 (0.011)
	12	0.020	0.021 (0.00081)	0.031 (0.0049)
Test	4	0.095	0.061 (0.0079)	0.078 (0.0070)
	8	0.038	0.035 (0.0024)	0.053 (0.0089)
	12	0.037	0.035 (0.0022)	0.046 (0.0050)

TABLE II
Mean–Squared Error (MSE) comparisons with $\sigma_n^2 = 0.025$.

starting position, while `Ramps` is not. In fact, the `Ramps` algorithm diverges from a *majority* of its starting positions. This is illustrated in Tables IV, V and VI which show the number of initial positions required to reach 10 stable solutions. The ratio of *total* starting positions to *successful* starting positions varies roughly from 7:1 to 15:1. The results are generally worse as the number of nodes increases, suggesting that the percentage of partitions leading to a stable solution decreases as the dominant structure in the function is removed from the residual. The results are also slightly better as the noise level is increased, which may be due to the increase in spurious local minima (additional attractors) that can sometimes accompany high noise situations. The fact that there is such a high percentage of unsuccessful starting positions not only slows the `Ramps` algorithm (because of the large number of restarts required), but degrades performance by making it harder to discover the best minima.

The results for `SweepingHinge` and `HingeDescent` are more comparable. On average, `HingeDescent` provides better performance in the no noise case, and in the beginning when very few nodes are present. But as the number of nodes is increased the performance of the two are roughly equivalent. This is especially true when noise is present. The advantage of `SweepingHinge` is that it is run only once. Its solution is not a function of the initial position as it is with `HingeDescent`. Because of the sweeping operation, `SweepingHinge` will always take longer than a single run of `HingeDescent`. But when `HingeDescent` is started from several initial positions the run times of the two algorithms are more comparable.

The second experiment in this section is designed to test the effect of dimensionality on the models produced by IIA/`SweepingHinge`. In section II we saw that under appropriate conditions the approximation error for sigmoidal basis function models is bounded by c_f/n, where c_f depends on f. It is trivial to show that this bound also applies to the empirical squared error. That is, when $f_{n,N}$ is

Data Set	Number of Nodes	MSE for Sweeping Hinge	Avg (Std) MSE for Hinge Descent	Avg (Std) MSE for RAMPS
Train	4	0.137	0.125 (0.0083)	0.156 (0.014)
	8	0.090	0.095 (0.0036)	0.119 (0.010)
	12	0.083	0.081 (0.0042)	0.104 (0.011)
Test	4	0.160	0.151 (0.0075)	0.170 (0.010)
	8	0.127	0.131 (0.0050)	0.144 (0.010)
	12	0.130	0.136 (0.0082)	0.143 (0.010)

TABLE III
Mean–Squared Error (MSE) comparisons with $\sigma_n^2 = 0.1$.

Number of Nodes	Average Number of Retries	Standard Deviation
4	87	70.5
8	127	97.7
12	155	109.3

TABLE IV
RETRIES NEEDED TO FIND 10 STABLE SOLUTIONS WITH RAMPS ($\sigma_n^2 = 0$).

Number of Nodes	Average Number of Retries	Standard Deviation
4	85.4	63.3
8	115.4	74.6
12	128.3	73.9

TABLE V
RETRIES NEEDED TO FIND 10 STABLE SOLUTIONS WITH RAMPS ($\sigma_n^2 = 0.025$).

Number of Nodes	Average Number of Retries	Standard Deviation
4	71.5	45.3
8	87.0	46.2
12	100.0	51.4

TABLE VI
RETRIES NEEDED TO FIND 10 STABLE SOLUTIONS WITH RAMPS ($\sigma_n^2 = 0.1$).

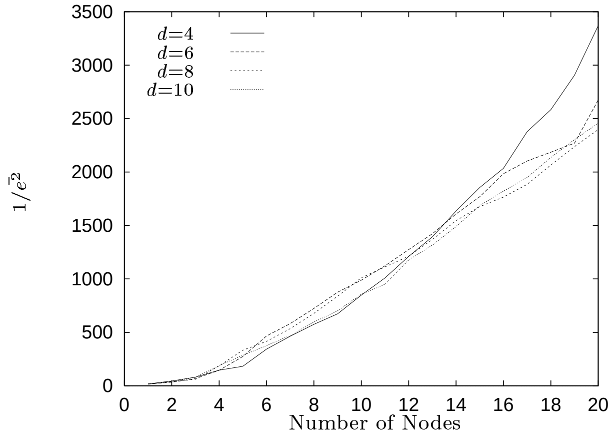

Fig. 18. Results for training data with $d = 4, 6, 8, 10$.

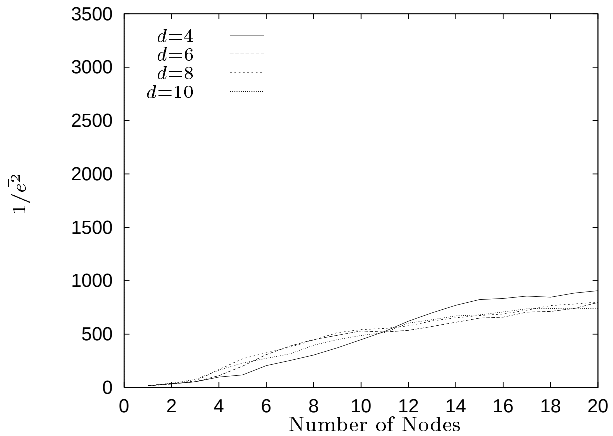

Fig. 19. Results for test data with $d = 4, 6, 8, 10$.

chosen to minimize the empirical squared error it can be shown that [28]

$$\bar{e^2} = \frac{1}{N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i) - f_{n,N}(\mathbf{x}_i)\right)^2 < \frac{c_f}{n} \tag{18}$$

Although we cannot guarantee a global minimum with the IIA/`SweepingHinge` algorithm, we will demonstrate that it is capable of producing results that satisfy (18).

The bound in (18) applies only to the error over the *training* data. The error over future data, i.e. the generalization error, includes both approximation error and *estimation* error (i.e. the error due to finite sample training). The estimation error generally prevents us from achieving (18) over an independent set of test data. This is demonstrated in the experiment below. Most importantly however, we demonstrate that both training and test errors are independent of dimension, giving empirical support for the claim that IIA/`SweepingHinge` algorithm can produce models that circumvent the curse of dimensionality.

The following experiment was adapted from [9]. The function $f(\mathbf{x}) = e^{-\|\mathbf{x}\|^2}$ is sampled at $100d$ points $\{\mathbf{x}_i\}$ such that $\|\mathbf{x}\| \leq 3$ and $\|\mathbf{x}\|$ is uniform on $[0, 3]$. The dimension d is varied from 4 to 10 (in steps of 2) and models of size 1 to 20 nodes are trained using the IIA/`SweepingHinge` algorithm. The number of samples traversed at each step of the sweep in `SweepingHinge` was set to $M = 10$. The average sum of squared error, $\bar{e^2}$, was computed for both the training data and an independent set of test data of size $200d$. Plots of $1/\bar{e^2}$ versus the number of nodes are shown in Figures 18 and 19 for the training and test data respectively. The curves in Figure 18 are clearly bounded below by a linear function of n (as suggested by inverting (18)). More importantly however, they show no significant dependence on the dimension d. The curves for the test data in Figure 19 (shown on the same scale) make it clear that the generalization error is larger than the training error (as expected). The asymptotic effect of the estimation error is noticeable in these curves as they start to "bend over" around $n = 10$ nodes. Again however, they show no real dependence on the dimension d.

The final experiment in this section compares the methods developed here with the results in [36], which examines several different nonparametric models on a variety of regression problems. The models in [36] include k–nearest neighbor (KNN), generalized memory–based learning (GMBL), projection pursuit regression (PPR), artificial neural networks (ANN), multivariate adaptive regression splines (MARS), and constrained topological mapping (CTM). The ANN model in [36] is a one–hidden layer network that is similar to the model in this paper except that it uses the smooth sigmoid activation function and is trained using a combination of conjugate gradient descent and simulated annealing. Several regression problems, which vary from two to six dimensions, are used in [36]. The comparison here is with the "high dimensional" (six dimensional) function given by

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + 0x_6$$

This function is sampled uniformly on $[-1, 1]^6$. Sets of size 25, 100 and 400 are used for training, and a test set of size 961 is used to measure generalization performance. The generalization measure is taken to be the normalized root mean square (NRMS) error, i.e. the standard deviation of the test set error divided by the standard deviation of the test set itself. Training sets are synthesized with three different signal–to–noise ratios (SNRs), ∞, 4, and 2. The noise is Gaussian with zero mean.

Both `SweepingHinge` and `Backpropagation` are used to produce models for this function. `Backpropagation` is trained for 1000 epochs with a learning rate of 0.001 (higher values lead to instabilities and/or excessive oscillations near the solution). In all cases the number of nodes is optimized to provide the best generalization performance.

The results are summarized in Table VII. Both `SweepingHinge` and `Backpropagation` provide inferior generalization performance in the small sample case. This is consistent with the ANN results in [36]. Also, in the noise–free case PPR is the most consistent at providing the best generalization [36], and is consistently better than both `SweepingHinge` and `Backpropagation` here. However, `SweepingHinge` performs very well in the medium and large sample cases when noise was present, and in fact generalizes better than all other methods in three of the four cases. `Backpropagation` does not produce superior results for any of the six trails, but outperforms `SweepingHinge` in the 400–sample noise–free case. Finally note that `SweepingHinge` tends to produce smaller models than `Backpropagation`. These results suggest that piecewise–linear sigmoidal networks trained with IIA/`SweepingHinge` are very competitive with other methods.

Number of Samples	SNR	Best NRMS from [36]	`SweepingHinge` NRMS (Nodes)	MLP with Backprop NRMS (Nodes)
25	∞	**0.496**	0.531 (2)	0.555 (4)
25	4	**0.546**	0.610 (2)	0.683 (4)
25	2	**0.677**	0.707 (3)	0.724 (3)
100	∞	**0.099**	0.183 (7)	0.240 (10)
100	4	0.319	**0.250** (6)	0.345 (13)
100	2	0.456	**0.349** (4)	0.551 (13)
400	∞	**0.039**	0.095 (10)	0.086 (11)
400	4	0.152	**0.150** (8)	0.166 (8)
400	2	**0.269**	0.271 (7)	0.337 (8)

TABLE VII
GENERALIZATION RESULTS FOR THE 6–DIMENSIONAL PROBLEM.

VI. SUMMARY

This paper has introduced a constructive algorithm for nonlinear function approximation that builds a 1–hidden layer neural network with piecewise linear sigmoidal nodes. Important properties of the algorithm include computational efficiency, guaranteed convergence in a finite number of steps, ease of use, solutions which are independent of initial conditions, a simple stopping criterion, good scaling properties and good fits on high dimensional data.

VII. ACKNOWLEDGMENTS

This work was inspired by the theoretical results of Barron [28] for sigmoidal networks as well as the "Hinging Hyperplanes" work of Breiman [9], and the "Ramps" work of Friedman and Breiman [35]. We would like to acknowledge many fruitful discussions with Chaouki Abdallah, Domingo Docampo, James Howse, Tom Caudell and Tim Draelos.

REFERENCES

[1] D. Barry, "Nonparametric bayesian regression," *Ann. Statist.*, vol. 14, pp. 934–953, 1986.

[2] G. Wahba, "Partial and interaction splines for the semiparametric estimation of functions of several variables," in *Proceedings of the 18th Symposium Amer. Statist. Assoc.*, T. Boardman, Ed., 1986, pp. 75–80.

[3] G. Wahba, *Spline models for observational data*, vol. 59, SIAM, 1990.

[4] T. Hastie and R. Tibshirani, "Generalized additive models (with discussion)," *Statist. Science*, vol. 1, pp. 297–318, 1986.

[5] J.H. Friedman and W. Stuetzle, "Projection pursuit regression," *J. Amer. Stat. Assoc.*, vol. 76, pp. 817–823, 1981.

[6] P.J. Huber, "Projection pursuit," *The Annuals of Statistics*, vol. 13, no. 2, pp. 435–475, 1985.

[7] J.H. Friedman, "Multivariate adaptive regression splines," Tech. Rep. 102, Labratory for Computational Statistics, Dept. of Statistics, Stanford University, 1990.

[8] L. Breiman, "The π method for estimating multivariate functions from noisy data," *Technometrics*, vol. 33, no. 2, pp. 125–160, 1991.

[9] L. Breiman, "Hinging hyperplanes for regression, classification and function approximation," *IEEE Transactions on Information Thoery*, vol. 39, no. 3, pp. 999–1013, 1993.

[10] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees*, Wadsworth & Brooks, Pacific Grove, CA, 1984.

[11] R. Battiti, "First- and second–order methods for learning: between steepest descent and newton's method," *Neural Computation*, vol. 4, no. 2, pp. 141–166, 1992.

[12] P.E. Gill, W. Murray, and M.H. Wright, *Practical optimization*, Academic Press, London ; New York, 1981.

[13] D.G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison–Wesley, 1984.

[14] E.B. Baum, "A proposal for more powerful learning algorithms," *Neural Computation*, vol. 1, no. 2, pp. 201–207, 1989.

[15] L.K. Jones, "A simple lemma on greedy approximation in hilbert space and convergence rates for projection pursuit regression and neural network training," *The Annals of Statistics*, vol. 20, pp. 608–613, 1992.

[16] M.R. Azimi-Sadjadi, S. Sheedvash, and F.O. Trujillo, "Recursive dynamic node creation in multilayer neural networks," *IEEE Transactions on Neural Networks*, vol. 4, no. 2, pp. 242–256, 1993.

[17] C.L.P. Chen, "A rapid supervised learning neural network for function interpolation and approximation," *IEEE Transactions on Neural Networks*, vol. 7, no. 5, pp. 1220–1229, 1996.

[18] S. Ergezinger and E. Thomsen, "An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 31–42, 1995.

[19] J.N. Hwang, S.R. Lay, M. Maechler, D. Martin, and J. Schimert, "Regression modeling in backpropagation and projection pursuit

learning," *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 342–353, 1994.

[20] R. Lengelle and T. Denoeux, "Training mlps layer by layer using an objective function for internal representations," *Neural Networks*, vol. 9, no. 1, pp. 83–98, 1996.

[21] J. Moody and P.J. Antsaklis, "The dependence identification neural network construction algorithm," *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 3–15, 1996.

[22] R. Setiono and L.C.K. Hui, "Use of a quasi–newton method in a feedforward neural network construction algorithm," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 273–277, 1995.

[23] Y. Shin and J. Ghosh, "Ridge polynomial networks," *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 610–622, 1995.

[24] M. Staley, "Learning with piece–wise linear networks," *International Journal of Neural Systems*, vol. 6, no. 1, pp. 43–59, 1995.

[25] Y. Zhao and C.G. Atkeson, "Implementing projection pursuit learning," *IEEE Transactions on Neural Networks*, vol. 7, no. 2, pp. 362–373, 1996.

[26] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[27] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

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

[29] A. Pinkus, *n–Widths in Approximation Theory*, Springer–Verlag, New York, 1985.

[30] A.T. Dingankar and I.W. Sandberg, "A note on error bounds for approximation in inner product spaces," *Circuits, Systems and Signal Processing*, vol. 15, no. 4, pp. 519–522, 1996.

[31] S. Sahni, "Computationally related problems," *SIAM J. of Computing*, vol. 3, no. 4, pp. 262–279, 1974.

[32] M. Bellare and P. Rogaway, "The complexity of approximating a nonlinear program," in *Complexity in numerical optimization*, P.M. Pardalos, Ed., pp. 16–32. World Scientific Pub. Co., 1993.

[33] M. Kozlov, S. Tarasov, and L. Hacijan, "Polynomial solvability of convex quadratic programming," *Dokl. Akad. Nauk SSSR*, vol. 248, pp. 1049–1051, 1979, Translated in *Soviet Math Dokl.* **20**, 1108–1111.

[34] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, vol. 10, Springer–Verlag, 1987.

[35] L. Breiman and J.H. Friedman, "Function approximation using ramps," in *Snowbird Workshop on Machines that Learn*, 1994.

[36] V. Cherkassky, D. Gehring, and F. Mulier, "Comparison of adaptive methods for function estimation from samples," *IEEE Transactions on Neural Networks*, vol. 7, no. 4, pp. 969–984, 1996.

Don Hush (S'78-M'86-SM'92) received his BSEE and MSEE degrees from Kansas State University, Manhattan, Kansas in 1980 and 1982 respectively, and his Ph.D. in engineering at the University of New Mexico, Albuquerque, New Mexico in 1986. From 1986 to 1987 he worked at Sandia National Laboratories in Albuquerque, New Mexico. He is currently an Associate Professor in the Electrical and Computer Engineering Department at the University of New Mexico. He is a Senior Member of the IEEE and a member of the International Neural Network Society. He is the coauthor of a 1990 Prentice-Hall text entitled Digital Signal Analysis, and has served as an Associate editor for the IEEE Tranactions on Neural Networks and the IEEE Signal Processing Magazine.

Bill Horne (S'85-M'87) for photograph and biography see p. 1070 of September 1997 issue of this TRANSACTIONS.