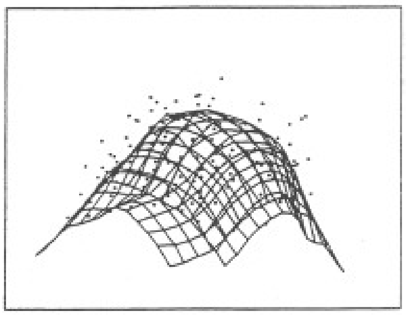
DEPARTMENT OF STATISTICS
Sequoia Hall
Stanford University
Stanford, CA 94305-4065

Fast MARS

Jerome H. Friedman

Technical Report No. 110
May 1993

Laboratory for Computational Statistics



**Department of Statistics
Stanford University**

FAST MARS

JEROME H. FRIEDMAN
Department of Statistics
and
Stanford Linear Accelerator Center
Stanford University

Abstract

Multivariate adaptive regression splines (MARS, Friedman (1991)) is a methodology for approximating functions of many input variables given the value of the function – perhaps contaminated with noise – at a collection of points in the input space. Although training times for this method tend to be much faster than feed-forward neural networks using back-propagation, it can still be fairly slow for large problems that require complex approximations (many units). This paper describes modifications to the original MARS algorithm that can dramatically increase its speed in these situations.

1.0. Introduction. Multivariate adaptive regression splines (MARS, Friedman (1991a, 1991b, 1991c)) have emerged as a viable competitor to other popular methods for supervised learning. The supervised learning problem is easily stated. One has a set of variables $\boldsymbol{x} = \{x_1, \cdots, x_p\}$ that are regarded as being simultaneous inputs to a system, and another set $\boldsymbol{y} = \{y_1, \cdots, y_q\}$ that are regarded as being the outputs. A relation of the form

$$y_j = f_j(x_1, \cdots, x_p) + \epsilon_j, \qquad i = 1, q \tag{1}$$

is presumed to exist between the inputs and outputs, where $\{f_j\}_1^q$ are single valued deterministic (target) functions of p variables, and $\{\epsilon_j\}_1^q$ are randomly varying quantities reflecting the fact that a simultaneous set of input values may not uniquely specify values for the outputs. The system under study may be responding to additional (hidden) inputs that are neither measured nor controlled. By convention the expected (average) values of each random component ϵ_j are taken to be zero, $E(\epsilon_j) = 0$. The goal of supervised learning is to obtain a useful approximation $\hat{f}_j$ to each (deterministic) function f_j in (1), using a set of examples (training data)

$$\{y_{i1}, \cdots, y_{iq}; x_{i1}, \cdots, x_{ip}\}_{i=1}^N \tag{2}$$

obtained by observing the system under study. Among the distinguishing characteristics of the MARS approach are generally fast training times, the ability to handle multivalued nonnumerical (categorical) input variables, and missing input values, in a natural way (Friedman (1991b)), and to produce approximating functions that have interpretative value. That is, they yield insight into the nature of the (multivariable) dependence of the outputs on the inputs (Friedman (1991a)).

2.0. MARS Algorithm. Mars produces approximations that take the form of an expansion on a set of basis functions $\{B_m(x_1, \cdots, x_p)\}_0^M$

$$\hat{f}(x_1, \cdots, x_p) = \sum_{m=0}^{M} a_m B_m(x_1, \cdots, x_p). \tag{3}$$

For a given set of basis functions, the coefficients are determined by a least squares fit of (3) to the training data

$$\{a_m\}_0^M = \underset{\{\alpha_m\}_0^M}{\operatorname{argmin}} \sum_{n=1}^{N} \left[y_n - \sum_{m=0}^{M} \alpha_m B_m(x_{n1}, \cdots, x_{np}) \right]^2. \tag{4}$$

In the MARS approach the basis functions are also adapted to the training data. They take the form

$$B_m(x_1, \cdots, x_p) = \prod_{k=1}^{K_m} b_{km}(x_{v(k,m)} \mid P_{km}) \tag{5}$$

which is a product of K_m (elementary) functions $b_{km}(\cdot)$, each of a single input variable $x_{v(k,m)}$, and characterized by a set of parameters P_{km}. If the input variable $x_{v(k,m)}$ takes on orderable numerical (real) values then

$$b_{km}(x \mid s,t) = [s(x-t)]_+ \tag{6}$$

where the subscript indicates the positive part of the argument

$$[z]_+ = \begin{cases} z, & \text{if } z > 0 \\ 0, & \text{otherwise} \end{cases} .$$

The parameters associated with (6) are the "knot" location $-\infty \le t \le \infty$ and the truncation sense $s = \pm 1$. That is, $P_{km} = (s_{km}, t_{km})$ in the corresponding factor (5). If $x_{v(k,m)}$ takes on a set of categorical values

$$x_{v(k,m)} \in \{c_1, \cdots, c_K\}, \tag{7}$$

where there is no order or distance relation among the values, then

$$b_{km}(x \mid A) = H(x \in A), \tag{8}$$

where A is a subset of the assumable values for x, $A \subset \{c_1, \cdots, c_K\}$, and $H(\eta)$ is a 0/1 valued function indicating the truth of its (logical) argument

$$H\eta) = \begin{cases} 1 & \text{if } \eta \text{ is true,} \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

In this case $P_{km} = A_{km}$, an enumerated subset of values for $x_{v(k,m)}$ (5).

The goal of the MARS algorithm (Friedman, 1991a,b) is to produce a good set of basis functions $\{B_m\}_0^M$ (3) (5) (6) (8), for approximating each output (target) function f_j (1), with feasible computation. This is accomplished through a forward/backward iterative approach. The forward part attempts to synthesize a (super) set of such basis functions. That is, a larger than optimal number is generated in a deliberate attempt to overfit the training sample (2). The backward iterative procedure then selectively deletes basis functions with the goal of producing the most generalizable approximation, where all basis functions are eligible for deletion. Motivation for this (forward/backward) approach is given in Friedman (1991a).

Basis function synthesis in the forward stepwise part of the algorithm proceeds as follows. The initial basis function set (at iteration $I = 0$) consists of the single (constant) function

$$B_0(\boldsymbol{x}) = 1. \tag{10}$$

Each successive iteration ($I \ge 1$) synthesizes two new basis functions of the form

$$B_{2I-1}(\boldsymbol{x}) = B_\ell(\boldsymbol{x}) b(x_v \mid P) \tag{11a}$$

$$B_{2I}(\boldsymbol{x}) = B_\ell(\boldsymbol{x}) b(x_v \mid \bar{P}), \tag{11b}$$

where $B_\ell(\boldsymbol{x})$ is one of the basis functions produced at an earlier iteration ($0 \leq \ell \leq 2I-2$), x_v is one of the input variables ($1 \leq v \leq p$), and $b(\cdot \mid P)$ takes the form of either (6) or (8), depending on the nature of x_v (real or categorical). The parameters $\bar{P}$ associated with (11b) are related to P, those associated with (11a). If x_v is real valued (6) then

$$P = (s, t) \text{ and } \bar{P} = (-s, t) \tag{12a}$$

whereas, if x_v is categorical (8)

$$P = A \text{ and } \bar{P} = \bar{A}, \tag{12b}$$

where $\bar{A}$ is the complement subset of values (of x_v) to those of A; that is,

$$H(x_v \in \bar{A}) = 1 - H(x_v \in A). \tag{13}$$

The two new basis functions (11) produced at each iteration are characterized by the parameters

$$\{\ell, v, P\}, \tag{14}$$

that is the particular previous ("parent") basis function $B_\ell(\boldsymbol{x})$, the particular input variable x_v that serves as the argument to the single variable function $b(\cdot \mid P)$, and the corresponding parameters P of that function. Values for these parameters are chosen to give the best (least-squares) fit of the resulting approximation to the training data,

$$(\ell^*, v^*, P^*) = \underset{\substack{\{a_i\}_0^{2I} \\ (\ell, v, P)}}{\operatorname{argmin}} \sum_{n=1}^{N} \Bigg[y_n - \sum_{i=0}^{2I-2} a_i B_i(\boldsymbol{x}_n) - a_{2I-1} B_\ell(\boldsymbol{x}_n) b(x_{vn} \mid P) - a_{2I} B_\ell(\boldsymbol{x}_n) b(x_{vn} \mid \bar{P}) \Bigg]^2. \tag{15}$$

These optimal parameter values (ℓ^*, v^*, P^*) are then used with (11) to create the two new basis functions at the Ith iteration. These are then included with the set of previously synthesized basis functions to serve as potential "parents" in future iterations. These iterations are continued until a (user specified) maximum number of basis functions, $M_{\max}$ are synthesized. [Heuristic rules for choosing $M_{\max}$ are given in Friedman (1991a).] The backward stepwise procedure is then applied to this final basis function set to selectively delete individual basis functions whose inclusion are judged (by a model selection criterion) to reduce generalizability. [See Friedman (1991a) for details.]

The (forward) algorithm develops basis functions of the form given by (5) in a hierarchical manner. New basis functions are created by multiplying one of the existing ones ("parent" basis function) by a (simple) function of a single variable. The final basis functions are thus each (tensor) products of factors, where each factor is a function of a single variable. The resulting approximation is obtained by a least-squares fit of the output on those basis functions that survive the backward elimination. Underlying motivations for this approach on approximational and statistical grounds, along with performance results on many real and simulated examples, are given in Friedman (1991a,b).

2.1. Computation. The forward stepwise part of the MARS procedure presents (by far) the greatest computational burden. At each step (iteration, I) the solution to (15) must be obtained. For a given set of parameters (14), the solution of (15) for the coefficients $\{a_i\}_0^{2I}$ can be obtained by standard least squares fitting methods with the dominant computation proportional to

$$C_{\ell sf} \sim NM^2 \tag{16}$$

where $M = 2I + 1$, the total number of coefficients (basis functions). The other parameters (14) enter nonquadratically in (15) so that numerical optimization methods must be used. The procedure employed is exhaustive search. That is, all possible (joint) parameter values are considered:

$$\begin{aligned} &0 \le \ell \le M - 2 \\ &1 \le v \le p, \end{aligned} \tag{17}$$

and for the factor parameters P,

$$s = \pm 1, \quad t \in \{x_{vn}\}_{n=1}^{N} \tag{18}$$

for x_v real (6), and

$$A \subset \{c_k\}_1^K \quad \text{(all subsets)} \tag{19}$$

for x_v categorical (7) (8) (9). This strategy requires on the order of pNM solutions to (15) for the coefficients $\{a_i\}_0^{M-1}$ (linear least-squares fits) each of which takes time proportional to (16). The computation associated with each iteration is therefore proportional to pN^2M^3. If there are $I_{\max}$ iterations, resulting in $M_{\max} = 2I_{\max} + 1$ total basis functions, then the total time to run the forward stepwise part of the MARS procedure is proportional to

$$C_{\max} = pN^2M_{\max}^4. \tag{20}$$

This computational requirement (20) grows rapidly with the training sample size N and super-rapidly with the complexity of the target function(s) (1) as characterized by the number of basis functions $M_{\max}$ required for an adequate approximation. For all but the smallest problems training times would be large, perhaps comparable to that required for feed forward networks trained through back propagation.

Friedman (1991a,b) takes advantage of special characteristics of this particular optimization problem to reduce computation. The time required for the individual linear least-squares fits (16) is (dramatically) reduced to

$$C^{*}_{\ell s f} \sim M \tag{21}$$

by considering the set of eligible (factor) parameter (18) (19) values in a special order. Least-squares updating formulae can then be used to obtain the solution (15) (for the coefficients) for a particular set of parameter values (18) (19) in time proportional to M, given the solution for the previously considered parameter values. This strategy reduces the total computation for the MARS algorithm (from (20)) to be proportional to

$$C^{*}_{\max} = pN M^{3}_{\max}. \tag{22}$$

This results in relatively fast training times for moderately sized problems ($pN \lesssim 20,000$, $M_{\max} \lesssim 50$), usuallly seconds to (a few) minutes on a typical engineering workstation (DECStation 5000/240).

Although the computational tricks leading to (21) (22) greatly expanded the size and scope of problems to which the MARS approach can be applied, there are still limitations for comfortable training times. The main problem lies with the cubic dependence on the number of basis functions (22). For very large problems (pN big) requiring complex approximations ($M_{\max}$ large) MARS training times can be prohibitive. This note describes a strategy for dramatically (further) reducing training times for MARS on these large complex problems. The central idea is to alter the strategy for optimizing with respect to ℓ and v (17) from an exhaustive search over all possible values at each iteration, to a much smaller set of values likely to contain the solution, as determined from information retained from previous iterations. The main result is to reduce the overall computation to be proportional to $M^{2}_{\max}$ (rather than $M^{3}_{\max}$) in (22).

3.0. Parent Priority Queue. A central ingredient in the MARS algorithm is to choose one of the existing basis functions $B_\ell(\boldsymbol{x})$ ($0 \le \ell \le 2I-2$) as the "parent" for producing two new "daughter" basis functions, $B_{2I-1}(\boldsymbol{x})$ and $B_{2I}(\boldsymbol{x})$ (11), at the Ith iteration. The one chosen to be the parent is the one that minimizes the lack-of-fit criterion

$$L_I(\ell) = \min_{\substack{\{a_i\}_0^{2I} \\ (v,P)}} \sum_{n=1}^{N} \left[y_n - \sum_{i=0}^{2I-2} a_i B_i(\boldsymbol{x}_n) - a_{2I-1} B_\ell(\boldsymbol{x}_n) b(x_{vn} \mid P) - a_{2I} B_\ell(\boldsymbol{x}_n) b(x_{vn} \mid \bar{P}) \right]^2, \tag{23}$$

or equivalently, maximizes the improvement-of-fit criterion

$$J_I(\ell) = \min_{\{a_i\}_0^{2I-2}} \sum_{n=1}^{N} \left[y_n - \sum_{i=0}^{2I-2} a_i B_i(\boldsymbol{x}_n) \right]^2 - L_I(\ell). \tag{24}$$

At each iteration, I, $J_I(\ell)$ is computed for all $0 \le \ell \le 2I-2$ and

$$\ell^* = \operatorname*{argmax}_{0 \le \ell \le 2I-2} J_I(\ell). \tag{25}$$

The corresponding (optimizing) parameters v^* and P^* are then used (with ℓ^*) to define the two new daughter basis functions through (11).

Except during the early iterations (I small), the approximation (3) does not change dramatically by the addition of two additional basis functions, at the next iteration. Therefore, there is likely to be a strong correlation between $J_I(\ell)$ and $J_{I+1}(\ell)$ $(0 \le \ell \le 2I-2)$, except perhaps for the one $J_I(\ell^*)$ (25) selected as the parent. More specifically, consider $\{J_I(\ell)\}_0^{2I-2}$ sorted in ascending order and let $R_I(\ell)$ be the rank of $J_I(\ell)$ in this sorted list,

$$R_I(\ell) = \operatorname*{rank}_{J_I(\ell)} \{J_I(k)\}_{k=0}^{2I-2}. \tag{26}$$

The basis function at the top of this list $(R_I(\ell^*) = 2I-2)$ is chosen to be the parent at the Ith iteration. At the $I+1$st iteration all of the $\{J_{I+1}(\ell)\}_0^{2I}$ are recalculated and (re)ranked yielding $\{R_{I+1}(\ell)\}_0^{2I}$. For I not small (later iterations), the rankings are not likely to be very different, $R_{I+1}(\ell) \simeq R_I(\ell)$, for $0 \le \ell \le 2I-2$, In particular, those basis functions with low rank $R_I(\ell)$ at the Ith iteration are most likely to also have low rank $R_{I+1}(\ell)$ at the $I+1$st iteration, and thus be very unlikely to be at the top of the list $(R_{I+1}(\ell) = 2I)$ to be chosen as the $I+1$st parent. This suggests that the search for the $I+1$st parent can be restricted to those basis functions with high rank $R_I(\ell) \ge 2I-2-K$ in the previous situation. Here $K << 2I-2$ is a specified cutoff value for limiting the search.

Consider a priority queue in which potential parent basis functions are ranked (26) by (24) at the Ith iteration. The length of this queue is $2I-2$. The basis function $B_{\ell^*}(\boldsymbol{x})$ the top of this queue $(R(\ell^*) = 2I-2)$ is taken to be the parent for creating the two new (daughter) basis functions $B_{2I-1}(\boldsymbol{x})$ and $B_{2I}(\boldsymbol{x})$ (11). These two new basis functions are placed at the top of this priority queue $(J_I(2I-1) = J_I(2I) = \infty)$, which now has length $2I$. The remaining basis functions thereby have their ranks reduced by two,

$$\{R_I(\ell) \leftarrow R_I(\ell) - 2\}_0^{2I-2}$$

retaining their relative ordering in this (larger) queue. At the $I+1$st iteration the improvement-of-fit (23) (24) is (re)calculated only for the top K members (basis functions) in this queue (i.e. $R_I(\ell) \ge 2I-K$). The improvements associated with the remaining basis functions, $R_I(\ell) < 2I-K$, are taken to be the same values as used at the previous (Ith) iteration. The entire queue is then reordered based on this set of improvement values and the resulting highest ranking basis function is selected as the parent for the $I+1$st iteration. The two new ($I+1$st) daughters are then placed at the top of this (now larger) queue for the next ($I+2$nd) iteration.

During the early iterations ($I \leq K+2$) all (previously created) basis functions are among the top K in the queue and the algorithm proceeds as before (Friedman, 1991a,b). In later iterations ($I > K+2$) parents with low (approximate) improvement potential ($R_I(\ell) < 2I-2-K$), as reflected by (low) improvement-of-fit values (23) (24) calculated at earlier iterations, do not have their improvements recalculated. Computation is thereby reduced by a factor of $K/(2I-2)$ at the Ith iteration. If the best potential parent (among all $2I-2$) lies among the top K members in the queue (as is likely) then there is no performance (accuracy) loss. Even if it does not, it is likely that a good competing parent (that would have been chosen at a later iteration) is among the top members and will be chosen. Its choice will tend to deflate its value for future iterations, thereby allowing the missed (best) parent to enter the top of the queue and be selected in a later iteration.

3.1. Aging. The priority queue strategy described in the previous section has a natural "aging" property. Namely, potential parents that are sent near the bottom of the queue at some iteration will tend to work their way back to the higher levels (over time) and be reevaluated. As iterations proceed the approximation (to the training data) becomes better. Thus, recomputed improvements (24) will tend to decreases since less improvement is possible. Improvements that have not been recomputed for a long time will tend to be larger (than if they were recomputed at the present iteration) reflecting the larger potential improvement possible then. This will cause them to work their way up in the priority queue. When they reach the top K in the queue they will be reevaluated. If they are still not good (low improvement measure) they will then revert to near the bottom of the queue and start working their way up again. Those parents near the top of the queue will have their improvements (24) reevaluated often (as long as they stay near the top) until they achieve a low improvement value.

It may be desirable to enhance the aging effect to make sure that some potential parents don't get lost forever. That is, they get assigned so low an improvement value at some stage, that they can never work their way back up in the queue. This can be accomplished by setting the priority for each basis function as

$$P_I(\ell) = R_I(\ell) + \beta(I - I_\ell) \tag{27}$$

where $R_I(\ell)$ (26) is the ranking based on improvement scores, I is the current iteration number, and I_ℓ is the iteration at which $J_{I_\ell}(\ell)$ was last recomputed. The coefficient β is the (artificial) "aging" factor. The top K parents as sorted on this priority (27) (rather than improvement) are then evaluated at the Ith iteration.

Larger values of β (27) result in low improvement parents rising faster in the (new) priority queue (27), providing insurance against a (possibly good) parent getting lost. A disadvantage of increasing β is a (slight) degradation of the general quality of the top K elements of the queue. In practice, setting $\beta = 1$ seems to represent a good trade-off.

4.0. Saving Solution Values. The parent priority queue strategy described above saves improvement values (23) (24) from previous iterations and recomputes them only if they are among the K largest in the queue. A by-product of calculating the improvement associated with a potential parent (23) (24) are the values of the parameters (v^*, P^*) yielding the minimum of (23). These are the values that would be used to construct the daughter basis functions (11) if that parent were chosen. The underlying assumption motivating the parent priority queue strategy is that the approximation (3) does not dramatically change from one iteration to the next, especially during the later iterations. Thus, one might suspect that the optimal parameters (v_ℓ^*, P_ℓ^*) associated with each parent $B_\ell(\boldsymbol{x})$ (23) might not substantially change either. This would motivate saving (v_ℓ^*, P_ℓ^*) when improvements (23) (24) are computed. These values would be used to compute corresponding improvements for $B_\ell(\boldsymbol{x})$ at later iterations, provided that they are not too much later when the approximation (3) has substantially changed. This can save considerable computation since (23) would be computed for only one set of values (v^*, P^*) rather than for all possible (joint) values.

It turns out that there is nothing to be gained by saving optimal values of the parameters P^* (11) (12). The least-squares updating formulae (Friedman, 1991a,b) permit the computation of optimal values (given ℓ and v (23)) nearly as fast as it takes to compute the lack-of-fit (23) for a given (single) set of values. It can make sense however to save the optimal (input) variable number v^* since evaluating (23) for a single value of v is p times faster than optimizing (calculating) it over all values $1 \leq v \leq p$.

The basic idea is to associate two (additional) integers (m_ℓ, v_ℓ) with each (parent) $B_\ell(\boldsymbol{x})$ in the priority queue. The first, m_ℓ, is the last iteration number at which the improvement (23) (24) for $B_\ell(\boldsymbol{x})$ was recomputed by optimizing over all values of $1 \leq v \leq p$. The second, $v_\ell = v^*$, is the corresponding optimizing input variable number. At the Ith iteration the top K basis functions in the parent priority queue have their improvements recalculated following the strategy outlined in Section 3.0 and 3.1. For each one, if

$$I - m_\ell > h \tag{28}$$

then the corresponding improvement is calculated from (23) (24) by a complete optimization over all input variables $(1 \leq v \leq p)$, as well as all parameter (P) values. Then

$$m_\ell \leftarrow I \text{ and } v_\ell \leftarrow v^* \tag{29}$$

where v^* is the optimizing input variable. If (28) is false, then the corresponding calculation is performed only over the values of P for input variable fixed at $v = v_\ell$, and neither m_ℓ nor v_ℓ are updated. This reduces the improvement computation (23) (24) by a factor of p (the number of input variables). The quantity h in (28) characterizes the frequency at which input variable optimizations are (re)performed. A value $h = 1$ will cause the optimal input variable to be recomputed (optimized) every time an improvement (23) (24) is recalculated. A value $h = 5$ will cause this complete optimization to be done for a parent basis function $B_\ell(x)$ only if it has been more than five iterations since it was last optimized for that basis function. Otherwise, the improvement (24) (24) is evaluated only for $v = v_\ell$ (23), reducing computation by a factor of p.

When two new (daughter) basis functions $B_{2I-1}(x)$ and $B_{2I}(x)$ are created (at the Ith iteration) m_{2I-1} and m_{2I} (28) are set to $-\infty$ so that v_{2I-1} and v_{2I} will be calculated by a complete optimization (23) at the next ($I+1$st) iteration. It is also prudent to set $M_{\ell^*} = -\infty$, where $B_{\ell^*}(x)$ was the selected parent at the Ith iteration, since its incorporation into the approximation (with its previously optimal v^*) will likely cause its optimal input variable to change for future iterations. Remember that these three basis functions (daughters and corresponding parent) are the first three elements of the priority queue for the next ($I + 1$st) iteration. Thus three complete optimizations (over the input variables $1 \leq v \leq p$) are performed at each iteration. For each of the remaining top K elements in the priority queue complete optimizations are performed with probability $1/h$ (28). The expected computation is reduced thereby from being proportional to Kp, to being proportional to $3p + (K-3)(1 - 1/h + p/h)$. This results in an (average) computational improvement ratio of

$$C(h, K) = \{3 + (K-3)[(1 - 1/h)/p + 1/h]\}/K. \tag{30}$$

5.0. Analysis. The computational reduction resulting from the strategies outlined in the preceding three sections is controlled by two parameters. They are K, the priority queue search depth, and h (28), the (inverse) frequency with which the optimization over input variables (23) is performed. Decreasing K and/or increasing h decreases computation. The trade-off for this benefit is a potentially less thorough optimization of (15) that could result in a less accurate approximation (3). The (possible) loss of accuracy issue is discussed in the next section. Here we derive formulae for the expected (relative) speed-up as a function of K and h, given the other parameters of the problem (N, p, and $M = M_{\max}$ (22)). These formulae can be used to predict running times for longer more thorough runs, after MARS has been run in a fast mode (K small, h large) for exploratory work. They also given an idea of the reduction in computation possible through these speed-up strategies.

We first perform a more careful analysis of the computation required by MARS in its most thorough mode ($K = \infty$, $h = 1$). Through the use of the least-squares updating formulae (Friedman, 1991a,b) the computation required to minimize (15) with respect to (only) the parameters P

is proportional to

$$C_P = Nm \tag{31}$$

where N is the training sample size and $m = 2I - 1$ is the number of parent basis functions at the Ith iteration. This minimization is performed for all input variables x_v, $1 \le v \le p$, and all parent basis functions $B_\ell(\boldsymbol{x})$, $0 \le \ell \le m-1$. Thus the time required for the Ith iteration is proportional to pNm^2. The total time for I_{tot} iterations is then (proportional to)

$$W_0 = pN[M(M+1)(2M+1)]/6 \tag{32}$$

where $M = 2I_{tot} - 1$.

We now analyze the the running time required (as a function of K and h) for the strategies presented in this paper (Sections 3.0–4.0). For the first $(K+2)/2$ iterations there are less than (or equal to) K parent basis functions in the queue, $m \le K$, so that all current members are examined. Also, we deliberately postpone invoking the input variable memory strategy (Section 4.0) for these same early iterations so that the first few (K) basis functions are constructed by a thorough optimization. Thus for $m = 2I - 2 \le K$ the running time W_1 is the same as that for the original strategy

$$W_1 = pN[K(K+1)(2K+1)]/6. \tag{33}$$

When there are more than K parents in the queue, $m > K$, the expected computation is proportional to $MNK \cdot C(h,K)$, from (30) (31), at each iteration for which $K < m \le M$. Thus, the computation for this ($m > K$) part of the procedure is

$$W_2 = NKC(h,K)[M(M+1) - K(K+1)]/2. \tag{34}$$

Therefore, the total computation is reduced by a factor of $R = (W_1 + W_2)/W_0$ (32) (33) (34) which becomes (after some algebra)

$$R = 3[CM(M+1) + (K+1)(2K^2/3 - (C - 1/3)K + 2C)]/[M(M+1)(2M+1)], \tag{35a}$$

where

$$C = K \cdot C(h,K) = 3 + (K-3)[1/h + (1-1/h)/p]. \tag{35b}$$

One sees from (35) that for fixed queue search depth K, and M becoming increasingly large ($M >> K$), that $R \sim 1/M$. Therefore, with this strategy, computation increases (with increasing total number of basis functions M) as M^2, rather than M^3 for the original MARS strategy (22) (32) [Friedman (1991a,b)]. Also, in this limit the computation increases linearly with K. From

(35b) one sees that the computational reduction associated with increasing h is most pronounced for p and/or K large, and in any case reaches a diminishing return as h increases.

In the original MARS algorithm, the computation associated with the forward stepwise basis function synthesis completely dominated the training time. The speed-up strategies discussed here can greatly reduce this part of computation (35). This, however, enhances the relative importance of other parts of the training procedure, that while insignificant before the speed-up, are unaffected by it. The computational reduction ratio (35) pertains only to the forward stepwise part of the procedure so that it gives an overly optimistic estimate of the speed-up of the entire procedure, especially when R (35) is very small. This is examined in the next section.

6.0. Simulation Studies. The strategies described herein for decreasing MARS training times are not purely algorithmic. That is, they do not perform the equivalent computation in less time, but rather they alter the computation (possibly) trading a (hopefully) small degree of accuracy (on the training data) for a big gain in speed. Training with different values of queue search depth K and input variable (inverse) updating frequency h, gives rise to different approximations (3). In order to choose appropriate values for these two parameters, it is important to know not only how they affect training time, but also how approximation accuracy is affected. These questions are examined in the context of two artificial examples. The result appears to be that while changing the values of K and h can have a dramatic effect on training computation times, approximation accuracy is largely unaffected over a wide range of their values.

6.1. Gaussian Wave Form Example. The input variables $\{x_i\}_1^p$ for this example are the relative amplitudes of a Gaussian function evaluated at $p = 20$ points on the real line

$$x_i = e^{-(1/2)(i-\mu)^2/\sigma^2}, \qquad 1 \le i \le 20. \tag{36a}$$

Each observation (exemplar) is characterized by a mean μ and standard deviation σ. These were generated to have a uniform distribution on

$$\begin{gathered} 0 \le \mu \le 20 \\ 0.1 \le \sigma \le 10.1. \end{gathered} \tag{36b}$$

The response (output) was taken to be the standard deviation σ (36). That is, the algorithm attempts to learn to calculate the standard deviation of a waveform (36a) given a sample of waveforms of different (unknown) means and (known) standard deviations (36b). A sample of $N = 400$ waveforms was used for training and the (maximum) number of basis functions was set to $M = 100$.

Table 1 shows results (rows) for various parameter (K, h) values (first two columns). The third column gives the corresponding accuracy (average squared-error) on the training sample. The fourth column gives the corresponding generalization accuracy obtained by computing the average squared-error over 5000 additional observations (36) that in no way participated in training. The fifth column gives the predicted computation reduction ratio (35) computed from the corresponding parameter values, whereas the last column gives the actual ratio of training execution times. The first line in Table 1 ("Time $(\infty, 1)$") gives the actual execution time for the first row (no speed-up) on a DecStation 5000/240.

Table 1

Gaussian waveform example with $N = 400$ training observations and $M = 100$ basis functions

Time $(\infty, 1) = 1975.0$ sec.

		$(\text{error})^2 \times 10^{-3}$		relative computation	
K	h	train	gen	estimated	actual
∞	1	.755	14.4	1	1
20	1	.622	4.17	.30	.33
20	5	.649	4.25	.11	.14
10	5	.690	4.62	.070	.10
10	10	.814	3.98	.06	.08
5	5	1.08	5.46	.05	.07
5	10	1.41	6.42	.05	.06

The last column of Table 1 indicates that fairly dramatic computational gains are possible through the strategies outlined in Sections 3.0–4.0. Column 5 shows that the estimated gain from (35) is (in this example) only slightly optimistic. The second and third columns show that the price (in accuracy) paid for these computational gains is (at most) minimal. In fact, the first row (no speed-up) gave (by far) the worst accuracy on both the training and generalization sample for some (as yet not understood) reason. With this (anamolous) exception, accuracy on the training sample degrades with increasing speed factor (K smaller, h larger) but not by very much. The effect on generalization error is seen to be even less.

6.2. A Robot Arm Example. In this example data are taken from a hypothetical robot arm free to move in three dimensions (x, y, z). Figure 1 shows a schematic diagram of the arm. It has two joints ($J1$, $J2$). The first $J1$ is fixed in location (at the origin) but has two rotational degrees-of-freedom; it can rotate the upper arm (a_1) in the $x - y$ plane around the origin (θ_1), and it can twist the upper arm (a_1) perpendicular to the direction a_1 is pointing (ϕ). The upper arm (a_1) is constrained to lie in the $x - y$ plane. The second (hinge) joint $J2$ can only rotate the forearm (a_2) around the location of $J2$ in the plane defined by the upper and forearms (θ_2). The lengths of both the upper and forearms are adjustable. The ($p = 5$) input variables were taken to be the lengths of the upper and forearm (ℓ_1, ℓ_2) respectively, and the three angles (θ_1, θ_2, ϕ). The response (output) was the distance from the origin ($J1$) to the end of the forearm (x, y, z) opposite to the joint ($J2$), the location of which is given by

$$\begin{aligned} x &= \ell_1 \cos\theta_1 - \ell_2 \cos(\theta_1 + \theta_2)\cos\phi \\ y &= \ell_1 \sin\theta_1 - \ell_2 \sin(\theta_1 + \theta_2)\cos\phi \\ z &= \ell_2 \sin\theta_2 \sin\phi. \end{aligned}$$

The (output) distance is then

$$d = (x^2 + y^2 + z^2)^{1/2}.$$

Training data was generated uniformly over the complete range of angles $\theta_1 \in [0, 2\pi]$, $\theta_2 \in [0, 2\pi]$, $\phi \in [-\pi/2, \pi/2]$, and arm lengths in the range $\ell_1 \in [0, 1]$ and $\ell_2 \in [0, 1]$. Two examples were run. The first was with $N = 400$ training samples and $M = 100$ basis functions. The results are shown in Table 2 (in the same format as Table 1). The second example had $N = 800$ training samples and $M = 200$ basis functions allowed. Both examples illustrate the dramatic computational gains possible by applying the speed-up strategies, with little (if any) apparent accuracy loss. The estimated computation reduction factor (35) is more optimistically biased here than in the first example (Table 1), but its relative values (among the smallest) give a reasonable reflection of (proportional) changes in computation for different parameter (K, h) values. Thus, after running in a high speed mode (K small, h large) one can obtain a fairly reliable estimate for the increase in computation required for running in slower modes for more thorough optimization. Although for examples presented here there appears to be little gain in accuracy by slower running, one may wish to verify this in any given situation.

Table 2

Robot arm example with $N = 400$ training observations and $M = 100$ basis functions

Time $(\infty, 1) = 235.7$ sec

		$(\text{error})^2 \times 10^{-3}$		relative computation	
K	h	train	gen	estimated	actual
∞	1	.720	6.07	1	1
20	1	.919	4.64	.30	.39
20	5	.974	6.08	.14	.20
20	10	.752	3.33	.12	.20
10	5	.963	5.33	.08	.15
10	10	.868	5.04	.08	.13
5	5	.998	7.44	.06	.11

Table 3

Robot arm example with $N = 800$ training observations and $M = 200$ basis functions

Time $(\infty, 1) = 3231.9$ sec

		$(\text{error})^2 \times 10^{-3}$		relative computation	
K	h	train	gen	estimated	actual
∞	1	.242	1.04	1	1
20	1	.223	1.03	.15	.23
20	5	.198	1.49	.07	.14
20	10	.327	1.27	.06	.11
10	5	.211	.813	.04	.11
10	10	.251	1.14	.04	.08
5	5	.416	1.23	.03	.06

Comparing Tables 2 and 3 one sees that he computation reduction ratio for $M = 200$ is roughly half that for $M = 100$, as is predicted by (35). As M is increased the corresponding computational gain with these speed-up strategies will also increase (roughly linearly) reflecting the fact that computation grows as M^2 (rather than M^3 with no speed-up).

All three tables reflect the diminishing computational return associated with increasing h, especially for K small, as predicted by (35). There seems to be little gain in increasing h beyond the value $h = 5$.

The error (squared) values displayed in the three tables do not exhibit a smooth increasing dependence with decreasing computation as might be expected by the fact that reduced computation implies a (possibly) less thorough optimization on the training sample. This is no surprise for generalization error since it is well known that closer fitting to the training data need not imply better generalization. However, this tendency is also exhibited (to a somewhat lesser degree) with the training error values. Upon reflection, this behavior is not unexpected. Changing the optimization procedure changes (usually slightly) the particular basis functions that are entered, at each iteration. Due to the forward stepwise (greedy) nature of the algorithm, basis functions constructed in later iterations depend on those synthesized earlier. It can happen that a (slightly) suboptimal basis function entered earlier can combine with a later one to produce a better result than if the optimal one had been entered earlier. This effect helps compensate for the (apparently very slight) degradation caused by the suboptimality, sometimes (as indicated by the Tables) completely overcoming it. This is more likely in situations characterized by very small approximation errors such as those considered here. It is clear from the Tables that there is very little (if any) accuracy degradation with the speed-up strategies.

It should be noted that these speed-up strategies can give rise to significant increases in accuracy when training computation rather than sample size is the critical resource. For the case of $N = 800$, $M = 200$ (Table 3) the resulting approximation error-squared is about five times less than for $N = 400$, $M = 100$ (Table 2). With the original MARS algorithm it would take about 14 times longer to run the larger problem, in order to gain this increase in accuracy. With the advent of the computation reduction strategies outlined in Sections 3.0–4.0, Table 3 (last row) shows that the larger problem can now be run in less time than the smaller one could have been run before.

7.0. Discussion.

The central idea underlying the strategies of Sections 3.0–4.0 is to install a memory into the MARS algorithm so that results from earlier iterations can help guide the optimization for later ones. The results of the simulated examples in Section 6 (Tables 1–3) indicate success, in that computation can be dramatically reduced with little or no apparent decrease in approximation accuracy. These examples represent purely approximational problems in that no error was added to the output. A set of input variable values completely specified the response output value. With output error there would be even less (relative) accuracy decrease since the irreducible error cannot be modeled (by definition) through any method.

Rogers (1991, 1992) proposed a quite different strategy for attempting to decrease computation for MARS (-like) approximations. He replaces the forward stepwise approach to basis function synthesis by a genetic optimization algorithm (Holland, 1975). No relative computation times are given, but for one problem a reduction by about a factor of ten in the number of (linear) least-squares fits is reported for the genetic approach as compared to the *original* MARS algorithm. However (as noted by Rogers) full least-squares fits are needed with the genetic approach, each requiring computation proportional to NM^2 (16). With MARS the least-squares updating formulae enable this to be reduced to be proportional to M (21). For the example cited by Rogers (1992) ($N = 200$, $M \simeq 20$), the computation for each MARS least-squares fit is thereby reduced on the order of a factor of 10^3. Thus, it would appear (from the information reported) that the original MARS algorithm (Friedman, 1991a,b), even without the speed increases developed here, is (considerably) faster than the genetic approach.

References.

Friedman, J. H. (1991a). Multivariate adaptive regression splines (with discussion). *Annals of Statistics*, **19**, 1–141.

Friedman, J. H. (1991b). Estimating functions of mixed ordinal and categorical variables using adaptive splines. Department of Statistics, Stanford University, Technical Report LCS 108.

Friedman, J. H. (1991c). Adaptive spline networks. In *Advances in Neural Information Processing Systems*, **3**, Morgan Kaufmann, San Mateo, CA.

Holland, J. (1975). *Adaptation in Artificial and Neural Systems.* University of Michigan Press, Ann Arbor, MI.

Rogers, D. (1991). G/SPLINES: A hybrid of Friedman's multivariate adaptive splines (MARS) algorithm and Holland's genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego, CA.

Rogers, D. (1992). Data analysis using G/SPLINES. In *Advances in Neural Information Processing Systems*, **4**, 1088–1095, Morgan Kaufmann, San Mateo, CA.

Figure Caption:

Figure 1. Schematic diagram of the robot arm used in Section 6.2.

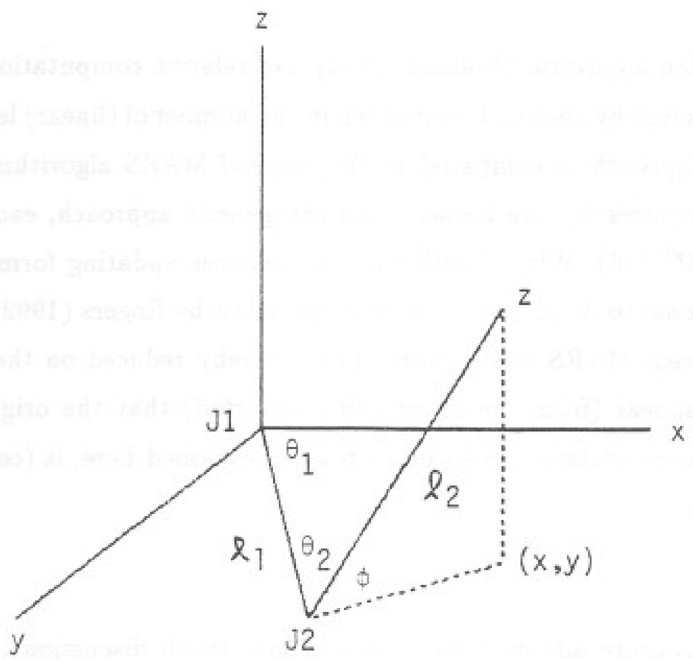


FIGURE 1