Selecting Concise Training Sets from Clean Data

Mark Plutowski and Halbert White

***Abstract*— We derive a method for selecting exemplars for training a multilayer feedfoward network architecture to estimate an unknown (deterministic) mapping from clean data, i.e., data measured either without error or with negligible error. Active data selection chooses from a given set of available examples a concise subset of training "exemplars." In practice, this amounts to incrementally growing the training set as necessary to achieve the desired level of accuracy. Our selection criterion does not depend on using a neural network estimator, thus it may be used for general purpose nonlinear regression using any statistical estimator.**

The objective is to minimize the data requirement of learning. In a particular sense, we are performing a kind of data compression, by selecting exemplars representative of the set of all available examples. Towards this end, we choose a criterion for selecting training examples that works well in conjunction with the criterion used for learning, here, least squares. We proceed sequentially, selecting an example that, when added to the previous set of training examples and learned, maximizes the decrement of network squared error over the input space. When dealing with clean data and deterministic relationships, we desire concise training sets that minimize the *Integrated Squared Bias* (ISB). We use the ISB to derive a selection criterion for evaluating individual training examples, the ΔISB, that we maximize to select new exemplars.

We conclude with graphical illustrations of the method, and demonstrate its use during network training. Several benefits are apparent for practical use in a variety of applications. Experimental results indicate that training upon exemplars selected in this fashion can save computation in general purpose use as well.

I. Introduction and Overview

A. Statement of the Problem

THE learning task is to train mutlilayer networks to estimate a deterministic function from clean data. We use a gradient descent learning rule to modify the network weights. Rather than using all of the training examples, we utilize information from a partially trained network to select a concise subset of exemplars.

If this approach turns out to be generally applicable, several benefits are immediately apparent for a wide range of situations. First, if the set of available training examples is too large to contain in working memory, our method provides a feasible method of selecting a concise subset of exemplars representative (in a precise sense) of the entire set. Second, our derivation of the selection criterion does not depend on using a neural network estimator, so our method may be used for any statistical estimator of a deterministic function. Third, the method could be used in data-driven learning techniques where the model complexity required to achieve functional interpolation is directly related to the number of training examples. For example, Platt allocates the basis functions at a rate directly associated with the number of exemplars, which are chosen randomly from the set of candidates [35]. Our principled way of choosing the exemplars could make Platt's algorithm more accurate or efficient by maximizing the information content of each exemplar, resulting in a smaller network.

Manuscript received February 26, 1991; revised March 29, 1992.

M. Plutowski is with the Department of Computer Science and Engineering, University of California, San Diego, CA.

H. White is with the Institute for Neural Computation and the Department of Economics, University of California, San Diego, CA.

IEEE Log Number 9200941

Maximizing the information density of the training set may have the side benefit of making training more reliable and efficient. Second order techniques that require evaluation of the entire training set at each iteration may consequently be made more efficient, e.g., methods involving the Hessian matrix and conjugate gradient techniques. Currently, heuristics are often employed with conjugate gradient algorithms for the purpose of reducing the number of examples that need to be considered at each iteration [32]. Applications that require a set of estimators to be trained repeatedly upon a set of data may also be made more efficient, once a concise set of exemplars is obtained. For example, some approaches to complexity regularization require several networks to be trained upon the same data set.

We demonstrate that our method selects concise training sets, and that it can be used to reduce the cost of network training. The method works well on networks with multiple inputs and outputs. Theoretically, it should work on problems of any dimension. However, experience in high-dimensional domains is required to determine whether the benefits of our technique outweigh the overhead of the selection process. We have confirmed this conjecture on small problems, showing a reduction in the number of floating point operations required to train a network on several learning tasks. In fact, we found that once a concise set of exemplars is found, the cost of retraining another network to fit the entire set of candidates can be done at a fraction of the cost of training over all the examples. We can also grow the network while simultaneously growing the training set, again resulting in a reduction of floating point operations on the problems we tried.

B. Approach

Our approach applies to the noiseless learning task exclusively. We choose a criterion for evaluating sets of training examples appropriate for use with the squared error performance criterion. It is intractable to select a set of exemplars that minimize this criterion directly, but it is feasible to maximize its decrement while growing the training set. Therefore, we sequentially add exemplars to the training set, training the network in between selection of successive exemplars. We derive a criterion for evaluating individual examples that allows us to select exemplars maximizing the decrement in squared error that would result from adding the new exemplar to the training set and training upon the resulting training set.

C. Previous Work

The problem of efficiently selecting data points for use in learning has been studied extensively. The problem has been referred to variously as "active learning," "sequential design," or "query-based learning." In a loose sense, we can think of a concise set of exemplars as analogous to the notion of "extreme points" employed by Cover for classification tasks [14]. In computational learning theory, deduction programs can make queries to an oracle [40], [20], [21], [29]. In the connectionist literature, there is a related approach in which a network is trained by *selective attention* [2].

Active data selection assumes that the model is allow to play an active role in either generating new exemplars or selecting exemplars from available examples. In the first approach (often called *query learning*), we are allowed to select the input values where future data will be gathered. In this case this task is to minimize the number of exemplars one has to gather, or to corroborate information crucial to the estimate. This is desirable when data measurements are costly, or when there are critical gaps in the data due to sampling variation. The second approach is the one we address in this paper, and we refer to it as *active exemplar selection*. This is concerned with selecting a concise subset of the available examples for use as training exemplars. This approach is concerned with minimizing the number of exemplars one has to train upon, given that many are available. Alternatively, we may view this approach as selecting a concise subset of exemplars representative of the entire set of available "candidate" training examples. This is also desirable when dealing with very abundant data, or when using data-driven learning methods. Note that active exemplar selection may utilize information about the targets since this information is available in the set of candidate examples, whereas query learning may select new exemplars based upon the input information only.

Active exemplar selection is inspired by, and in some respects, is similar to, query learning. Therefore, we provide a brief overview of query learning with references for the interested reader. Some approaches to query learning use a partially trained network to determine *regions of uncertainty* in the environment and then query the oracle for values of the function in these regions [12]. Others use statistical sampling techniques to ensure that the environment is sampled sufficiently often in regions of the input space where the data is deemed to have high utility [43]. Hwang *et al.* present a query-based approach for neural network learning of classification tasks that generates exemplars in the vicinity of classification boundaries [25], [24]. Baum proposes another query-based learning algorithm for iteratively seeking out classification boundaries for which a convergence proof is supplied [5]. MacKay uses a principled approach from a Bayesian perspective to obtain an information-based query strategy for training classification networks. The performance of the resulting technique corroborates the intuitive strategy of concentrating queries near classification boundaries [31].

Using queries for the more general task of estimating an unknown mapping from real-valued, noisy data has been studied extensively in the statistics and econometrics literature, in the theory of *response surface methodology*, *optimal experiments*, and *sequential design* [17], [33], [15], [9], [28]. These techniques typically query in regions where the network estimate has high uncertainty. Stringent conditions are typically required to ensure that second order derivative information (or approximation thereof) is well-behaved. It may be necessary to ensure that the model is the correct one for the task, which is translated into our situation as meaning that the network must have the appropriate complexity [30]. A significant amount of data may be required to "prime" the techniques [15]. Edge effects can also be a problem, since the borders of the input domain can tend to the regions of high uncertainty [30], [17].

If it is known that the function to be learned is reasonably smooth, a query method exists for a kernal-based smoothing approach. Faraway uses the integrated mean squared error (IMSE) criterion to sequentially query the environment for training examples for a nonparametric regression [15]. For more on the IMSE criterion the interested reader may refer to [17], [3], [9], [28], [13], [34], [39]. Fedorov obtains several querying methods for performing interpolation and extrapolation from noisy real-valued data using a correctly specified model, including a method for obtaining new data which is maximally informative with respect to comparing competing models.

Our approach is inspired by the work of Fedorov and Faraway, where the IMSE criterion is used to evaluate training sets. Our sampling criterion, the ISB, is derived directly from the IMSE as a special case appropriate when dealing with clean data and deterministic relationships [36]. Note that the query-based methods discussed above can be used in conjunction with our technique, to obtain more data from the environment once the current set has been learned.

II. Definition of the Learning Task

We consider the learning task of determining the relationship between two (possibly random) vectors X and Y, where X takes values in $\boldsymbol{X} \subset \Re^r$, and Y takes values in $\boldsymbol{Y} \subset \Re^s$, for integers r and s. We refer to $\boldsymbol{X}$ as the *input space*. For concreteness we take X to represent a vector of input variables and Y a vector of target variables. For simplicity, let $s = 1$. The learning task is thus one of training a neural network with r inputs and one output.

We adhere to the framework provided by White [41]. We draw n observations on $X, x^n = (x_1, \cdots, x_n)$, along with n corresponding observations on $Y, y^n = (y_1, \cdots, y_n)$. For convenience let $Z = (X', Y')'$, and $z_i = (x_i', y_i')'$, where the prime denotes transpose. We denote our *sample*, or, *training set*, as $z^n = (z_1, \cdots, z_n)$. We suppose that the measurement process could be continued indefinitely, to obtain a sequence of observations $\{z_i\} = \{z_i, i = 1, 2, \cdots\}$, as a realization of the random sequence $\{Z_i = (X_i', Y_i')'\}$, where X_i and Y_i have the distributions of X and Y, respectively for all $i = 1, 2, \cdots$.

In our simulations we will have complete control in determining x_i. We define a (fixed) "environmental" probability law μ that describes the relative frequencies with which different values for x_i are set; $\mu : B(\Re^r) \to [0,1]$. $B(\Re^r)$ is the Borel σ-algebra generated by the open sets of $\Re^r$.

If y_i and x_i can be measured with perfect precision, and the relation between Y and X is deterministic, then there is an exact functional relationship between x_i and y_i such that $y_i = g(x_i)$ for some mapping $g : \Re^r \to \Re$. We also write $Y = g(X)$ and $Y_i = g(X_i)$. This is also the limit of the case in which the relation between X and Y is random but measurement at the sample points is repeated many times, and the measurement averaged. In this case g represents the average relationship, $g(X) = E(Y|X)$, the conditional expectation of Y given X.

Let f denote the output function of a neural network parameterized by weights w, where w is an element of a weight space $W \subset \Re^p$ (we assume W compact). Then, for some particular input vector x, and weight vector w, the network output is $f(x,w)$. We will measure network performance using squared error. Our objective is to find a set of weights $w^* \in \arg\min_w \int (g(x) - f(x,w))^2 \mu(dx)$,

III. Derivation of Method

A. Integrated Squared Bias

We consider a (measurable) nonlinear least-squares estimator

$$\hat{w}_n \in \arg\min_w \frac{1}{n} \sum_{i=1}^{n} (Y_i - f(X_i, w))^2.$$

Given appropriate conditions, $\hat{w}_n$ displays a strong form of stochastic convergence to optimal weights w^*, in that $\hat{w}_n \to w^*$ a.s.-P. [41, Theorem 1]. One of these conditions dictates that $\{Z_i\}$ be independent and identically distributed (i.i.d.) random variables. Here, we violate this condition. Given $z^n = (z_i, i = 1, \cdots, n)$, we choose x_{n+1} according to some specific criterion that will depend upon z^n. Therefore, the z_i are obtained from random variables that are neither independent, nor identically distributed. Nevertheless, we expect that if $\{x_i\}$ is properly chosen, then a similar convergence result can be established.

A criterion for selecting training examples should work well in conjunction with the criterion used for learning (here, squared error.) We take as a suitable choice of criterion the Integrated Squared Bias (ISB), defined below. The ISB is the special case of the Integrated Mean Squared Error (IMSE) appropriate for use on clean data [9], [28].

Define l_n as the learning rule giving the realization w_n of $\hat{w}_n$ as a function of the realizations x^n, y^n:

$$w_n = l_n(x^n, y^n) \in \arg\min_w \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i, w))^2.$$

Because we consider the case in which we learn from clean data, y^n is completely determined by x^n, as is w_n. Therefore, for any x^n the training sample becomes $z^n = (x^n, g(x^n))$, where we abuse notation by writing $g(x^n) = (g(x_1), g(x_2), \cdots, g(x_n))'$. Integrating the squared error over the input space gives the ISB:

$$\mathrm{ISB}(x^n) = \int (g(x) - f(x, l_n(x^n, g(x^n))))^2 \mu(dx), \quad (1)$$

where μ is an appropriately chosen measure on $(\boldsymbol{X}, B(\boldsymbol{X}))$.

B. Minimizing ISB for Nonlinear Least Squares Regression

For a sample of size n, the optimal training set satisfies $x_*^n = \arg\min_{x^n} \mathrm{ISB}(x^n)$. Finding such a training set is computationally impractical. Here we pursue a more modest goal. Given a training set x^n, we shall seek x_{n+1} that maximizes the decrement in ISB, $\mathrm{ISB}(x^n) - \mathrm{ISB}(x^{n+1}), x^{n+1} = (x^n, x_{n+1})$. Note that this property for x^{n+1} is a necessary condition for solving the computationally impractical global problem, but will not necessarily deliver the globally optimal solution. Nevertheless, this approach permits a computationally feasible and attractive method for sequential selection of exemplars.

Let $\breve{x}_{n+1}$ be a candidate input, and $\breve{x}^{n+1} = \{x^n, \breve{x}_{n+1}\}$. Let $\lambda_n(x^n) = l_n(x^n, g(x^n))$, (where $g(x^n)$ is as defined in the previous section), so that $w_n = \lambda_n(x^n)$, and put $\breve{w}_{n+1} = \lambda_{n+1}(\breve{x}^{n+1})$. Thus we seek to maximize

$$\begin{aligned}\mathrm{ISB}(x^n) - \mathrm{ISB}(\breve{x}^{n+1}) &= \beta^2(\lambda_n(x^n)) - \beta^2(\lambda_{n+1}(\breve{x}^{n+1})) \\ &= \beta^2(w_n) - \beta^2(\breve{w}_{n+1}) \quad (2)\end{aligned}$$

where $\beta^2(w) = \int [g(x) - f(x,_w)]^2 \mu(dx)$. (The variable λ is introduced to make explicit the role of x^n in the noiseless environment; the variable β^2 is introduced to allow a change of argument emphasizing the role of the network weights.) We could pick x^{n+1} to maximize (2) directly, albeit at great expense. Fortunately, a much simpler procedure can be obtained by exploiting certain properties of the learning algorithm.

To proceed, we take a first order Taylor expansion of $\beta^2(\breve{w}_{n+1})$ around $\beta^2(w_n)$, yielding

$$\begin{aligned}&\beta^2(w_n) - \beta^2(\breve{w}_{n+1}) \\ &\quad \approx (w_n - \breve{w}_{n+1})' \\ &\quad \cdot \nabla_w \int (g(x) - f(x, w_n))^2 \mu(dx).\end{aligned}$$

By Bartle, ([4], Corollary 5.9), the gradient and integral can be interchanged to yield

$$\begin{aligned}\nabla_w \int (g(x) - f(w, w_n))^2 \mu(dx) = -2 \int \nabla_w f(x, w_n)(g(x) \\ - f(x, w_n))\mu(dx) \quad (3)\end{aligned}$$

provided in addition to the compactness of W that $\partial f / \partial w$ exists on $X \times W$, and that $f(x,\cdot)$ and the elements of $\nabla_w f(x,\cdot)$ are dominated by square integrable functions of x only. We have already assumed implicitly that g is square integrable. Consequently,

$$\begin{aligned}\beta^2(w_n) - \beta^2(\breve{w}_{n+1}) \approx -2(w_n - \breve{w}_{n+1})' \int \nabla_w f(x, w_n) \\ \cdot (g(x) - f(x, w_n))\mu(dx). \quad (4)\end{aligned}$$

Equation (4) provides a computationally simpler estimate of (2). For each successive selection of an example from a set of candidates, the integra in (4) need be computed once, and this can be approximated by summing over the set of candidate examples. Then, to evaluate each individual example in the set of candidates, we compute $(w_n - \check{w}_{n+1})$, and take its inner product with the vector quantity obtained from the summation over the set of candidates.

To calculate (4), we need to estimate $(w_n - \check{w}_{n+1})$, given $\check{x}_{n+1}$. When n is large, it is generally true that $\check{w}_{n+1}$ is in a neighborhood of w_n. The particular weight update we assume satisfies this requirement. For example, as described in [39], we may use the Gauss–Newton weight update:

$$w_n - \check{w}_{n+1} \approx - \left[\frac{1}{n+1} H(\check{x}^{n+1}, w_n)\right]^{-1} \frac{1}{n+1} \cdot \nabla_w f(\check{x}^{n+1}, w_n)\left(g(\check{x}^{n+1}) - f(\check{x}^{n+1}, w_n)\right). \tag{5}$$

Here,

- $f(\check{x}^{n+1}, w_n) = (f(x_1, w_n), f(x_2, w_n), \cdots, f(x_n, w_n), f(\check{x}_{n+1}, w_n))'$.
- $\nabla_w f(\check{x}^{n+1}, w_n) = (\nabla_w f(x_1, w_n), \cdots, \nabla_w f(x_n, w_n), \nabla_w f(\check{x}_{n+1}, w_n))$, a matrix with p rows and $n+1$ columns.
- $H(\check{x}^{n+1}, w_n)^{-1} = (\nabla_w f(\check{x}^{n+1}, w_n) \nabla_w f(\check{x}^{n+1}, w_n)')^{-1}$ acts as an approximation to the so-called Hessian matrix of second derivative information [17], [37], [39].

As an aside, note that the Gauss–Newton weight update we use here may be replaced. For example, we might use a steepest descent update, which would save computation at the cost of reduced accuracy. On the other hand, we could use a Newton weight update, which would provide higher accuracy but would require the Hessian matrix of second derivatives.

The estimate of $w_n - \check{w}_{n+1}$, (5), can be simplified by noting that for $n = 1, 2, \cdots, 1/n \sum_{i=1}^{n} \nabla_w f(x_i, w_n)(g(x_i) - f(x_i, w_n)) = 0$. This is by definition of w_n, which is optimal for x^n. (Note that this remains a valid assumption when w_n is approximated by a locally optimal estimate $\tilde{w}_n$ obtained by gradient descent.) As a result,

$$\nabla_w f(\check{x}^{n+1}, w_n)\left(g(\check{x}^{n+1}) - f(\check{x}^{n+1}, w_n)\right) = \nabla_w f(\check{x}_{n+1}, w_n)(g(\check{x}_{n+1}) - f(\check{x}_{n+1}, w_n)).$$

We also employ the approximation

$$\left[\frac{1}{n} H(x^n, w_n)\right]^{-1} \approx \left[\frac{1}{n+1} H(\check{x}^{n+1}, \check{w}_{n+1})\right]^{-1}.$$

Collecting the above approximations together, define our estimate of $w_n - \check{w}_{n+1}$ to be

$$\Delta\check{w}_{n+1} \equiv - \left[\frac{1}{n} H(x^n, w_n)\right]^{-1} \frac{1}{n+1} \nabla_w f(\check{x}_{n+1}, w_n) \cdot (g(\check{x}_{n+1}) - f(\check{x}_{n+1}, w_n)) \tag{6}$$

giving as one implementation of (4)

$$\beta^2(w_n) - \beta^2(\check{w}_{n+1}) \approx 2\Delta\check{w}'_{n+1} \int \nabla_w f(x, w_n)(g(x) - f(x, w_n))\mu(dx). \tag{7}$$

For use in comparing candidate examples with each other, note that the constant factors of (6) may be eliminated, giving

$$\Delta\check{w}_{n+1} = [H(x^n, w_n)]^{-1} \nabla_w f(\check{x}_{n+1}, w_n)(g(\check{x}_{n+1}) - f(\check{x}_{n+1}, w_n)) \tag{8}$$

Ignoring the constant factor in (7) and combining with (8) leads to the criterion that we propose for selecting the next training example,

$$\Delta\text{ISB}(\check{x}_{n+1}|x^n) = (g(\check{x}_{n+1}) - f(\check{x}_{n+1}, w_n))' \cdot \nabla_w f(\check{x}_{n+1}, w_n)'[H(x^n, w_n)]^{-1} \cdot \int \nabla_w f(x, w_n)(g(x) - f(x, w_n))\mu(dx). \tag{9}$$

In practice the distribution μ is unknown, but can be well approximated by the empirical distribution of the candidate examples. This means that the integral appearing in (9) is approximated by an average over all the candidate examples. This average need be computed only once. The training example x_{n+1} is selected so that

$$x_{n+1} \in \arg\max_{x_{n+1}} \Delta\text{ISB}(\check{x}_{n+1}|x^n). \tag{10}$$

Although we have arrived at this criterion by making use of approximations valid for large n, this criterion has an appealing interpretation apart from large sample considerations. Specifically, ΔISB can be interpreted directly as the square of the weighted Euclidean distance between the gradient of squared error with respect to network weights integrated over all candidate examples, and the gradient of squared error with respect to network weights for a single candidate example. (The matrix H^{-1} performs the weighting.) Thus, picking x_{n+1} to maximize ΔISB can be viewed as picking the single example having individual error gradient most highly correlated with error gradient of the entire set of examples. Learning with this example is therefore likely to be especially informative, relative to other available examples. The ΔISB criterion thus possesses heuristic appeal in training sets of any size. As n becomes large, the derivation above establishes that ΔISB approaches $\text{ISB}(x^n) - \text{ISB}(\check{x}^{n+1})$, providing further support for its use.

IV. Implementation

We now consider the practical use of the ΔISB criterion (9) for evaluating examples for training a one-hidden layer feedforward network. We have available a set of N examples from which to select training examples. Call this set $\boldsymbol{x}^N$, the set of candidate examples. Suppose we have trained the network using a subset of n examples x^n. Given x^n, we evaluate $\Delta\text{ISB}(\check{x}_{n+1}|x^n)$ for all $\check{x}_{n+1} \in \boldsymbol{x}^N$. For simplicity, we begin by putting $n = 1$ and find x_1 such that $x^1 = \{x_1\} = \arg\min_{x^1} \text{ISB}(x^1)$. This is computationally appealing, since fitting a single example by a neural network can be accomplished efficiently. Given *a priori* knowledge about the class of functions that we can expect the network learning rule to utilize to fit a single example, it is possible to calculate the optimal location of x_1 by employing summary statistics over the set of available examples. A single example is fit perfectly with a constant function (see Fig. 3(a) and 6(a).) It is then straightforward to choose the single example minimizing $\Delta\mathrm{ISB}(x^1)$. Calculate $m = (1/N)\sum_{i=1}^{n} g(x_i), x_i \in \boldsymbol{x}^N$, and select $x_1 = \arg\min_{x\in\boldsymbol{x}^N}(g(x) - m)^2$.

Given x^1, we proceed to grow the training set by successive optimization of $\Delta\mathrm{ISB}$. For $n = 1, 2, \cdots$, we then choose

$$x_{n+1} = \arg\max_{x} \Delta\mathrm{ISB}(x|x^n). \tag{11}$$

In growing the training set, a number of practical matters must be addressed. Because we are trying to learn a deterministic mapping, it is appropriate to specify a tolerance tol_N representing the desired accuracy desired for the fit over all of the candidates as measured by

$$\mathrm{rmse}_N = \left[\frac{1}{n}\sum_{x\in\boldsymbol{x}^N}(g(x) - f(x, w_n))^2\right]^{1/2}.$$

We desire $\mathrm{rmse}_N \le \mathrm{tol}_N$. Clearly, before selecting a new exemplar, we should always check whether $\mathrm{rmse}_N \le \mathrm{tol}_N$, and quit if it does. We refer to a set of exemplars that give this condition as "sufficient."

If this condition does not hold, we must determine when the current exemplars have been fit well enough to select a new one, as the underlying theory and practical experience suggest that, for a given network complexity, the selection criterion will work best if the fit is optimal for the current set of exemplars. Optimality for the given network complexity is ensured either by the finding that rmse_n is sufficiently small, where

$$\mathrm{rmse}_n = \left[\frac{1}{n}\sum_{x\in x^n}(g(x) - f(x, w_n))^2\right]^{1/2}$$

or by finding that repeated optimization attempts do not give further improvement in rmse_n. In implementing the requirement that rmse_n be sufficiently small, we specify that $\mathrm{rmse}_n \le \mathrm{tol}_N$ when selecting x_{n+1}. This requirement follows from the observation that if the given model cannot adequately fit the training points, then it is unlikely it will adequately fit the entire set of candidates, since $x^n \subset \boldsymbol{x}^N$.

Our experience indicates that fitting x^n to an even closer tolerance can result in a more concise set of exemplars, i.e., a sufficient set of smaller size. When the minimal number of exemplars is desired, rmse_n should be stringently minimized.

If a somewhat larger than minimal set of exemplars is acceptable, then it may not necessary to obtain the globally minimal value for rmse_n before selecting a new exemplar. The task of seeking the most concise training set can require more total overhead (which includes training as well as selection cost) than for another sufficient but larger set. Relaxing the tolerance on rmse_n allows a new exemplar to be selected sooner. This reduces the cost of training over the exemplars, and can thus reduce total overhead as well, even though more exemplars may need to be selected to comprise a sufficient set. We regulate this by using a variable tol_n to determine when rmse_n is sufficiently small to select a new exemplar.

A problem that can arise during this process is that an exemplar may be selected that is close to a previous choice, in the sense that both input and target values are close together. This can indicate that optimization in the prior exemplars was not complete (i.e., rmse_n can be reduced further) but also is observed to occur when network complexity is insufficient to achieve $\mathrm{rmse}_n \le \mathrm{tol}_N$. Such exemplars too close to a previous selection are easy to detect, and in such cases we simply discard the new exemplar and continue training on the current set. To avoid the cost of repetitively selecting a new exemplar only to reject it, we modify tol_n adaptively during training by reducing it when a selection is rejected. This also makes the method less sensitive to initial setting of tol_n, in that we may set it to a larger value initially, say, $\mathrm{tol}_n = \mathrm{tol}_N$, and then adaptively reduce it during training as necessary. Practical implementation of exemplar rejection is based on selecting a small number d. In our experiments the input and output were scaled equivalently, so we let $d = O(\mathrm{tol}_N)$. If the input vector a new exemplar is within Euclidean distance d of a previous exemplar in input space, we compute the distance between their output values, and if their output values are closer than tol_N, we reject the new exemplar. This permits us to keep exemplars in regions of the input space where the output varies rapidly with the input.

If the learning algorithm reaches a local minima with $\mathrm{tol}_N < \mathrm{rmse}_n$, then either the learning rule needs to continue its search within the current weight space in order to obtain a better optimum, or network complexity is insufficient to fit the current set of exemplars. Therefore, at this point we either restart network training or modify network complexity and continue.

If the learning algorithm reaches a local minima with $\mathrm{tol}_n < \mathrm{rmse}_n \le \mathrm{tol}_N < \mathrm{rmse}_N$, we attempt to add a new exemplar, even though $\mathrm{rmse}_n \le \mathrm{tol}_n$ has not been met, since we have found that the method will often select an informative exemplar anyway (i.e., one not too close to previous choices.) If the new exemplar is rejected, we either restart network training or modify network complexity and continue.

It may be difficult or expensive to determine whether a given network can fit a set of exemplars within tolerance. Although we may not know in general when an estimate is sufficiently close to a globally optimal solution, in particular circumstances we may employ capacity results for neural networks to determine whether this is the case. (cf., Vapnik-Chervonenkis dimension [1], [6].) For example, assume that each unit employs the usual sigmoid squashing function. If h is the number of hidden units in the neural network, then, given any set of $n < h$ examples, there exist weights w_n such that $\sum_{i=1}^{n}(g(x_i) - f(x_i, w_n))^2 = 0$. [23, Theorem 2.5]. Thus, in this case we know in advance that a set of weights exists that can fit the training examples exactly.

To summarize, we now present an algorithm for implementing the $\Delta\mathrm{ISB}$ criterion that embodies each of the considerations addressed above. The general algorithm begins with a network of arbitrary complexity, and increases the complexity appropriately as the exemplars become too difficult for the network to fit. In our experiments we always begin with a network consisting of a single hidden unit. We use multistarts to determine when the network is having difficulty fitting the data, by restarting the network a number of times on the data before increasing network complexity. Let *restarts* count the number of restarts, and let *max* be the maximum number of restarts we allow before increasing network complexity. Note that we reset *restarts* after adding a new exemplar, since each new exemplar results in a new training set, $x^n, n = 1, 2, \cdots$.

1) Begin by selecting x_1 minimizing $\text{ISB}(x^1)$, and set *restarts* $= 0$.
2) Continue training until one of the following occurs and take the actions listed, in sequence:
   (a) $\text{rmse}_n < \text{tol}_N$
   - If $\text{rmse}_N < \text{tol}_N$, quit.
   - Select $x_{n+1} = \arg\max_x \Delta\text{ISB}(x|x^n)$.
   - Check whether x_{n+1} is too close to previous exemplars. If x_{n+1} is rejected, reduce tol_n and continue training.
   - Otherwise, append x_{n+1} to x^n, set *restarts* $= 0$ and continue training.

   (b) $\nabla_w f(x^n, w_n)(g(x^n) - f(x^n, w_n)) = 0$ and $\text{rmse}_n \le \text{tol}_N$:
   - If $\text{rmse}_N \le \text{tol}_N$, quit.
   - Otherwise, select $x_{n+1} = \arg\max_x \Delta\text{ISB}(x|x^n)$.
   - Check whether x_{n+1} is too close to previous exemplars. If x_{n+1} is not rejected, append it to x^n, set *restarts* $= 0$, and continue training.
   - Otherwise, if *restarts* $<$ max, restart the network, increment *restarts*, and continue training.
   - Otherwise, increase network complexity, set *restarts* $= 0$, and continue training.

   (c) $\nabla_w f(x^n, w_n)(g(x^n) - f(x^n, w_n)) = 0$ and $\text{rmse}_n > \text{tol}_N$:
   - If *restarts* $<$ max, restart the network, increment *restarts*, and continue training.
   - Otherwise, increase network complexity, set *restarts* $= 0$, and continue training.

This algorithm could be modified to utilize more sophisticated ways of regulating network complexity. It is useful in its present form for many practical situations, and is sufficient for demonstrating the technique.

V. Demonstrating the Technique

A. Illustrating the Behavior of ΔISB

Here we wish to convey the qualitative behavior of the ΔISB criterion. We illustrate this using the network illustrated in Fig. 2 to obtain the target function $g(x)$ illustrated in Fig. 1(a). This is a one-dimensional map with upper and lower plateaus at about 0.2 and 0.8, respectively, and steep drop off. The drop off is shifted left of center, so that the lower plateau is more extensive than the upper plateau. The diagram illustrates a feedforward network with 1 input, 1 output, and 1 hidden unit. Each unit employs for its transfer function the usual sigmoidal output activation $o(a) = (1 + \exp(-a))^{-1}$, where a is the weighted sum of the inputs to the unit. The arcs are labelled by real values, denoting the weights. The values of the weights in this figure give a set of weights for which $f(x, w) = g(x)$. This set of weights is $w^* = (-20.0, 70.0, 1.5, -3.0)$. Note the size of weights w_1 and w_2 relative to w_3 and w_4. This is a manifestation of the region in g of high first derivative, which the network function fits by employing large magnitude weights on the hidden unit. In what follows, we will illustrate the ΔISB for three different network functions obtained by perturbing the weights of this network slightly. For clarity of illustration, we will assume all inputs are equally likely, so that μ is the uniform distribution.

In Fig. 1(b)-(d), we modify one weight at a time to obtain network functions slightly perturbed from $g(x)$. In each figure, the network function so obtained is given by the dashed line. This has the incidental benefit of illustrating the role each weight plays in the network mapping. For example, in Fig. 1(b) weight w_3 was perturbed, by putting $w_3 = 1.9$. This results in an upward shifting of the entire mapping. The dotted line gives the plot of ΔISB, normalized to allow it to be plotted along with $f(x, w)$ and g. Plotting f and $g(x)$ together allows a comparison of the two functions, and provides a visualization as to how the ΔISB criterion varies as the difference between f and g. Fig. 1(b) corroborates the *intuition that the sampling criterion indicates the potential of a candidate* example for reducing squared error over the input space. Although both plateaus are shifted upwards an equal distance, a training example chosen from the region of input space corresponding to the lower plateau is weighted more heavily than one chosen from the upper plateau. Presumably, that is because an example from the lower plateau might result in bringing the lower plateau of f back in line with g, and this would be better in comparison with choosing an example from the upper plateau, which covers a smaller region of the input space. To see where the next exemplar would be selected by (10), observe on the coordinate axis the location of the maximum value of the Delta ISB curve.

Fig. 1(c) and (d) illustrate that the criterion gives high value to candidate examples lying in regions where more information is necessary. The criterion displays an intuitively pleasing behavior in these figures, but is not obvious that the criterion will always act in accordance to our immediate intuition in more general settings. For example, the criterion may place high value on examples in regions where network error is substantially lower than the network error over other regions. This can occur by the effect of μ which essentially weights each example according to the environmental probability law. It may also occur due to the form of the basis functions. This is because small changes in the network function in one region resulting from the attempt to fit a particular example can have a global impact upon the network function far away from the new exemplar. This is illustrated in the next sections, where we next apply the ΔISB criterion in a sequential manner during network training.

B. Training a Fixed Size Network on Actively Selected Exemplars

We investigate what sequence of examples will be chosen by the criterion while learning the mapping of Fig. 1(a). Examples will be chosen from a set of candidates, $\boldsymbol{x}^N$ generated by the network of Fig. 2. Here, $N = 200$. Between selection

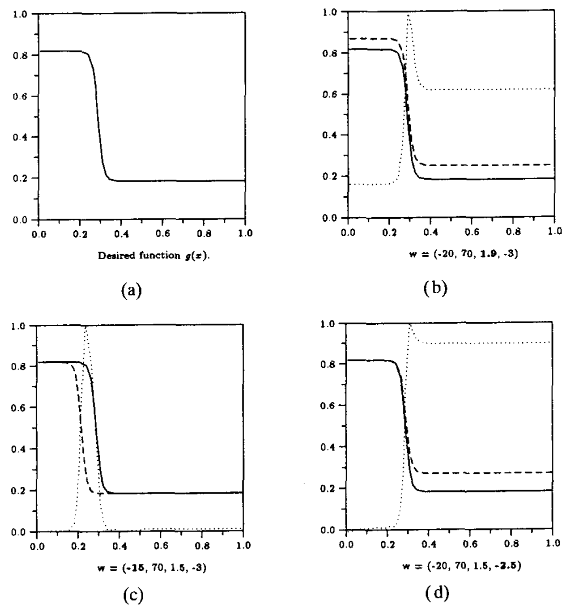


Fig. 1. (a) Shows the desired function, $g(x)$, here a scaled and translated sigmoid generated by the network of Fig. 2 with weights $w = (-20, 70, 1.5, -3.0)$. The next three figures illustrate the ΔISB upon networks obtained by perturbing these weights. The perturbed weight vector is given at the bottom of each figure, with the perturbed weight in bold. The network function $f(x, w)$ is given by the dashed line, $g(x)$ is given by the solid line, and the ΔISB, normalized for plotting purposes, is given by the dotted line.

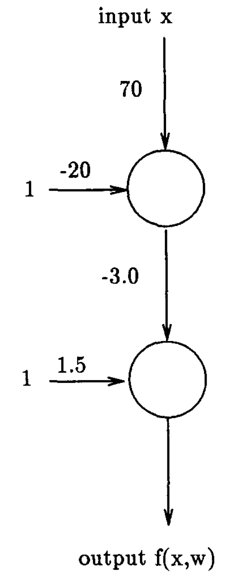


Fig. 2. A simple one hidden layer neural network.

of successive examples, the network function is optimized over the training set, x^n. Subsequent to optimization over x^n, obtaining w_n, $\Delta\text{ISB}(x|x^n)$ is calculated for each $x \in \boldsymbol{x}^N$, and x_{n+1} chosen from $\{x \in \max_x \Delta\text{ISB}(x|x^n)\}$. The training set is updated, obtaining $x^{n+1} = \{x^n, x_{n+1}\}$. We proceed with optimizing $f(x, w)$ over x^{n+1} to obtain w_{n+1}, and so forth.

Discussion of some implementation details is in order. We use the conjugate gradient algorithm to set the network weights, in order to make the computational comparisons more fair. We discuss this in more detail below in Section V-B-1). As our selection criterion, we use the implementation of (10) given by (9). In the interest of computational simplicity, we modified the equation for the purpose of demonstrating the technique. The particular gradient descent algorithm we used (conjugate gradient) does not calculate $H(x^n, w_n)$, so we used an approximation to it obtained by calculating only the diagonal elements of $H(x^n, w_n)$. This has the additional benefit of resulting in a matrix that is guaranteed to be positive definite, and inexpensive to invert. When the number of current exemplars was less than the number of parameters in the network, we set $H(x^n, w_n)$ to the identify matrix to avoid the situation where elements of the diagonal matrix become large due to ill-conditioning. Although we recommend using $H(x^n, w_n)$ whenever possible, we note that the performance of the ΔISB criterion did not seem overly sensitive to the approximations we made here.

We followed the algorithm given above in Section IV. We began with a training set consisting of a single example. In order to select x_1, we calculate the average values of the outputs over the candidate examples, as discussed above, and selected as the first exemplar the candidate x with $g(x)$ closest to this value (see Fig. 3(a)). The initial training set is indicated in the figure by a marker superimposed upon plot of the desired function, $g(x)$. We initialized the network by randomizing the weights to values in $[-0.3, 0.3]$, and trained the network over x^1, using a value of $\text{tol}_N = 0.01$. The resulting network $f(\cdot, w_1)$, gives a constant function passing through $g(x_1)$. After obtaining w_1 on x^1, we computed $\Delta\text{ISB}(x|x^1)$ for $x \in \boldsymbol{x}^N$. ΔISB is normalized by its maximum absolute value, so that it may be plotted along with f and g. Note that $\Delta\text{ISB}(x|x^1)$ may take on negative values.

The next exemplar selected by (10) is located by the maximum value of the ΔISB curve. This exemplar is appended to the initial training set x^1 to give the second training set, x^2. Fig. 3(b) shows the result of training on x^2, the two examples in x^2 being indicated, as before, by markers superimposed upon $g(x)$. Note how training on x^2 resulting in improvement along the entire lower plateau. After obtaining w_2, ΔISB was maximized at a point x_3 near the top of the steep portion of g. Note that this point is not one for which network error was maximal. Hence, ΔISB need not coincide with the heuristic of training on examples according to maximal error. We expect this to be the case for the squared error learning rule we have assumed. After training on x_3, the network function shows only a slight improvement, but this is enough to allow the selection of the final exemplar which provides enough information to complete the learning task. The final exemplar was also selected on the steep portion, near, but not quite on the upper plateau. However, due to the sigmoidal basis functions used in the network, the information provided by x_3 and x_4 is sufficient for the network to learn the upper plateau, even though no exemplars are located in that region. The result is a network function $f(\cdot, w_4)$ within tol_N of g.

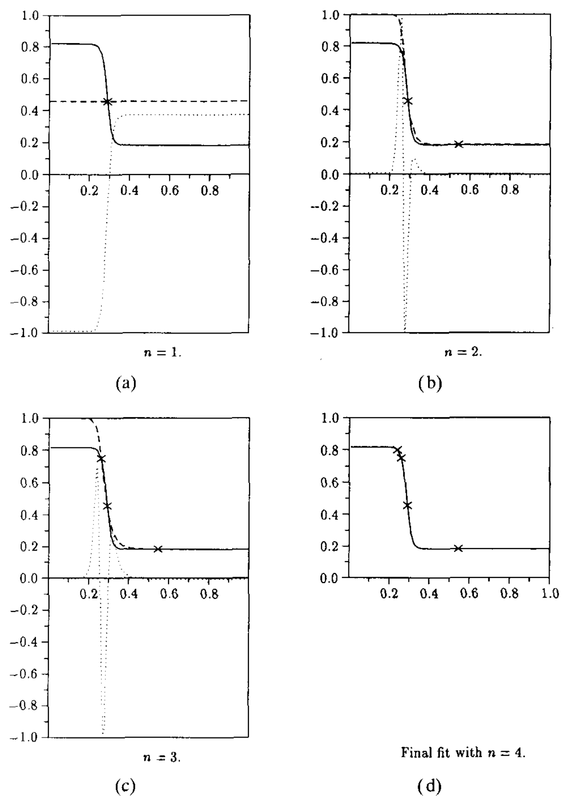


Fig. 3. These figures illustrate the sequential selection of exemplars during learning. The solid line gives $g(x)$, here, the scaled and translated sigmoid of the previous figure. The dashed line gives $f(x, w_n), n = 1, 2, 3, 4$, and the dotted line gives the $\Delta\text{ISB}(x|x^n)$ for $n = 1, 2, 3$. The exemplars are given by the cross hatch marks on the solid line. For each n, the new exemplar x_{n+1} is given by the lowest valued x maximizing $\Delta\text{ISB}(x|x^n)$.

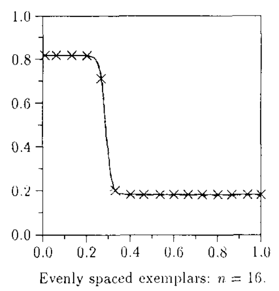


Fig. 4. Selecting uniformly spaced exemplars required 16 exemplars to fit the sigmoidal function.

An alternative selection method can be obtained by selecting exemplars uniformly distributed in the input space. We chose a sequence of training sets in this fashion and found that this particular learning task required 16 evenly spaced exemplars in order to fit the function within the desired tolerance. The set of exemplars selected in this fashion is displayed along with the resulting trained network function in Fig. 4. Note the sparsity of exemplars on the steep portion of the curve, and the relative density of exemplars along the plateaus.

1) Computational cost comparison: Here we illustrate how active selection may save computation in general purpose use. We compare it with the baseline approach of training over all available examples. Before reporting the results, we first discuss some relevant implementation details.

We measured computational cost in terms of floating point operations. We counted every multiply, divide, and add used during every phase of network training. It suffices to report only on the mutiplies here, since the number of adds and divides have been proportional to the multiplies in all of our experiments.

We compare the computational cost of the algorithm in Section IV with the cost of using all of the candidate examples. The question is whether or not the overhead due to exemplar selection will be compensated for by savings due to a lower cost of processing a concise training set. We also keep track of the cost of selecting exemplars, in order to determine the percentage of the total cost that is required in overhead. This allows us to analyze whether or not it is important to try to reduce the overhead of the selection process. However, the bottom line is whether or not the total cost of fitting the given set of candidates may be reduced by active selection.

The ΔISB method requires a close fit over the exemplars. This may be obtained using any kind of gradient descent, such as a stochastic approximation method (e.g., backpropagation), or steepest descent. We tried using a steepest descent algorithm in previous experiments, and found that several restarts were typically necessary for the learning rule to converge to weights near the optimal. This made the experiments costly to run, and comparisons difficult. Using steepest descent upon actively selected exemplars sometimes required restarts, but the cost of each restart was much less, due to the smaller training set size. Our intuitive judgment is that the ΔISB method would be far cheaper in this case, as we expect our technique to become more cost effective as the number of restarts increases. In order to make more reliable comparisons and to obtain the best baseline we chose to use conjugate gradient, which requires far fewer restarts to find a good set of weights. This largely avoided the case in which a large number of restarts introduced an experimental bias favoring our technique. While we do not do so here, it would be interesting to explore whether active selection could make steepest descent competitive with more sophisticated techniques by reducing the cost of restarts.

Now we present the results of comparing the cost of active selection with the baseline on the learning task illustrated in Fig. 3(a)-(d). In this case, the network is static, needing only one hidden unit. We used $d = \text{tol}_N = 0.01$, and $\text{tol}_n = (1/2)\text{tol}_N$. We experimented with candidate sets of size 200 and 500, to illustrate how the number of available examples affects the cost benefit. For each size candidate set we compared the total cost of training using active selection with the total cost of training on all of the candidates. This gives four cases. We evaluated five runs for each case. To ensure each training run started from the same initial conditions, we generated five sets of random initial weights, using the same set of five initial weights to initialize the training runs in each of the four cases.

TABLE I

Shifted sigmoid, $N = 200$.

Run	Baseline cost	Active selection cost	Active selection overhead	n
1.	2.94	0.47 (16.0%)	46.0%	4
2.	2.63	0.58 (22.1%)	35.6%	4
3.	2.02	0.58 (28.7%)	35.6%	4
4.	9.64	0.46 (4.8%)	46.8%	4
5.	1.85	0.45 (24.3%)	57.5%	5
Med:	**2.63**	**0.47 (17.9%)**	**46.0%**	

TABLE II

Shifted sigmoid, $N = 500$.

Run	Baseline cost	Active selection cost	Active selection overhead	n
1.	6.67	0.82 (12.3%)	58.5%	4
2.	8.80	0.89 (10.1%)	53.4%	4
3.	7.34	0.76 (10.4%)	64.4%	4
4.	15.65	0.95 (6.1%)	63.6%	4
5.	6.92	0.95 (13.7%)	55.0%	4
Med:	**7.34**	**0.76 (10.4%)**	**63.6%**	

As indicated by the results of Tables I and II the method selected sets of 4 exemplars on all but one of the runs, indicating the consistent performance of the method. Table I shows the comparison of the cost of learning the shifted sigmoid function using 200 candidate training examples. The "baseline" column gives the cost of training over all 200 examples. the next set of columns pertain to active exemplar selection. The total cost of training is given along with the ratio of this cost with the baseline cost in parenthesis. The "overhead" gives the portion of the total cost due to exemplar selection. The number of exemplars selected is given by n. cost is measured in millions of multiplies. The bottom row gives the median values. Table II shows the comparison of the cost of learning the shifted sigmoid function using 500 candidate training examples. The "baseline" column gives the cost of training over all 500 examples. The next set of columns pertain to active exemplar selection. The total cost of training is given next. The ratio of this cost to the baseline cost is given in parentheses. The "overhead" is the portion of the total cost due to exemplar selection. The number of exemplars selected is given by n. Cost is measured in measured in millions of multiplies. The bottom row gives the median values. The average overhead of the selection method comprised 58.8% of the total cost on the runs using 200 candidates, and 43.5% on the runs using 500 candidates. Observe that if the overhead due to exemplar selection is subtracted from the total cost, we see that the cost due to training is significantly less than the cost of training over all available examples, corroborating our intuition that concise training sets can reduce computation. The important question, however, is whether or not training by active selection will reduce computation when the overhead is included in the total cost. As indicated on the results, this was indeed the case for this experiment. The median cost of the ΔISB method required 17.9% of the median baseline cost on 200 candidates, and only 10.4% of the baseline on 500 candidates. Note that when N is increased from 200 to 500, the median cost of active selection grew less than the median baseline cost, which more than doubled.

C. Active Selection and Complexity Regularization

We now illustrate how the sampling criterion behaves upon a network which does not necessarily have the correct number of hidden units to fit the desired function exactly. We generated a mapping using the 3-hidden unit feedforward network with

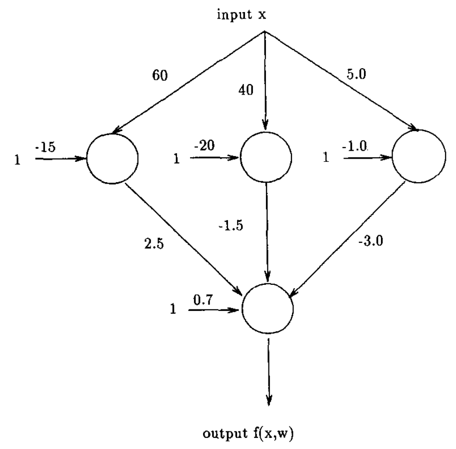


Fig. 5. Network 2.

the connectivity and weight values shown in Fig. 5, (which we refer to as Network 2) using the same type of sigmoidal units used in Network 1. Network 2 generates the skewed unimodal mapping $g(x)$ illustrated in Fig. 6(a). This choice of $g(x)$ gives a function difficult for a network of this particular architecture to fit with less than three hidden units. For the first demonstration we used Network 2 to generate $N = 200$ candidate examples uniformly at random over the input space.

The training regime is the same as for the experiment reported in the previous section, except that here we are allowed to grow the network. The network initially has one hidden unit. The goal is to fit the candidate examples to within a rmse_Ntolerance of $\text{tol}_N = 0.01$. We used $\text{tol}_n = 0.001$. As training progresses, the algorithm automatically modifies network complexity by adding another hidden unit when the network can no longer fit the exemplars. When the network reached a local minima without attaining $\text{rmse}_n \leq \text{tol}_N$, we restarted the network, randomizing the weights to small values. A maximum of 5 restarts was allowed before growing the network. Examining Fig. 6(c), we expect that this network could not fit the given x^3 with a single hidden unit, since the basis function used here is a sigmoidal function. Our complexity regularization technique detected this automatically, growing the network at the appropriate time. This also occurs when $n = 5$, (see Fig. 6(e)) where a third hidden unit was added to fit the 5 exemplars. The ΔISB method worked well along

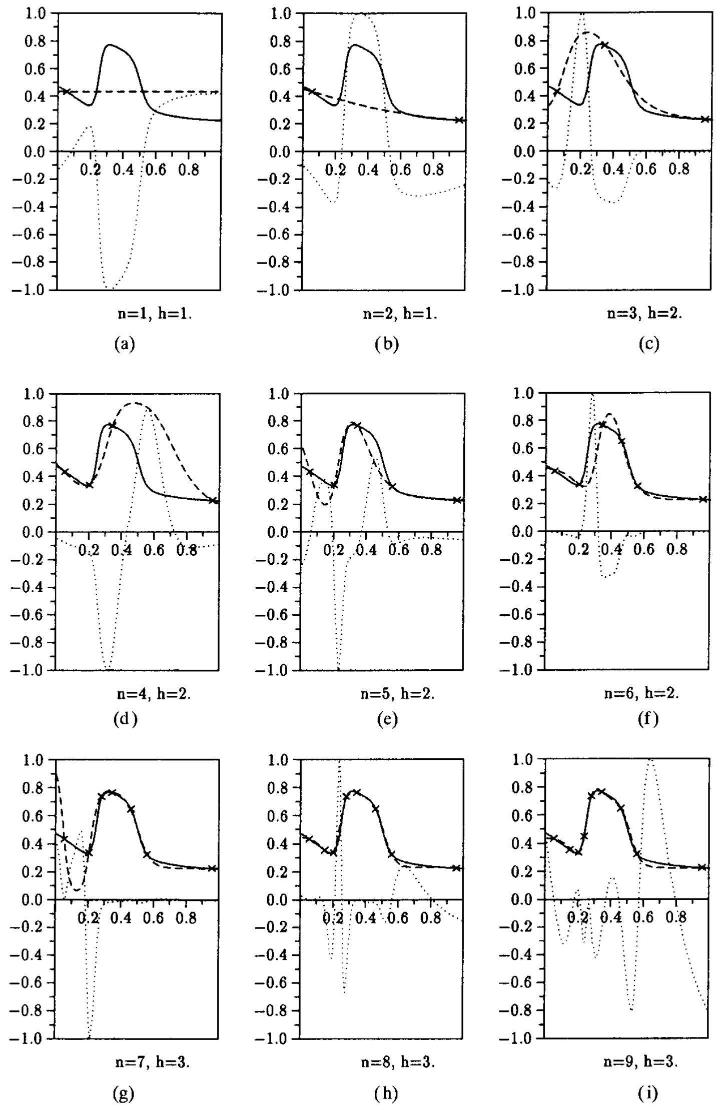


Fig. 6. The solid line shows the desired function, g, here a skewed hump. The exemplars are given by markers superimposed upon g. For each n, The network function $f(\cdot, w_n)$ is given by the dashed line. For each n, x_{n+1} is given by the smallest x maximizing ΔISB, which is given by the dotted line.

with this complexity regularization technique. The experiment concludes with the close fit illustrated in Fig. 7.

We compared this selection method with uniformly spaced exemplars. This task required 31 uniformly spaced exemplars in order to fit the candidates within the desired tolerance. The set of exemplars selected in this fashion is displayed along with the resulting trained network function in Fig. 8. This figure illustrates how the uniformly spaced distribution required a close spacing of exemplars in order to sample rapidly changing portions of the function sufficiently well.

1) Cost comparison: We now compare the cost of active selection with the baseline of training over all of the candidate examples upon the skewed hump learning task. This task required choosing the correct sized network for the task, so we applied the network growing method described above to

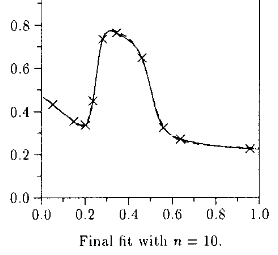


Fig. 7. Final result of active selection on the skewed hump. 10 exemplars were required overall.

the baseline method. We started each method with a single hidden unit, and added an extra unit if the exemplars have not

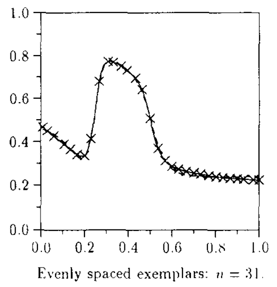


Fig. 8. Selecting uniformly spaced exemplars required 31 exemplars to fit the skewed hump.

TABLE III

Learning skewed hump						
	Baseline		Active selection			
N	cost	restarts	cost	restarts	overhead	n
200	495	10	55.5	10	2.5%	11
200	56.1	0	32.7	5	3.2%	10
500	286*	0	63.4	5	4.5%	11

Note*: the third baseline run entered a local minima without fitting x^N on the first try, and required a restart to converge; the number entered here is the cost of the second run.

been fit within tolerance after 10 restarts (see Table III). Table III compares the cost of active selection with the baseline of using all of the available examples. The task is to learn the skewed hump generated by network 2. The first column gives the number of candidates, N. The second pair of columns pertain to the baseline of training on all N candidates. Cost is measured in millions of multiplies. This task involves choosing the correct size network, so the maximum number of restarts is given. The next set of columns apply to active exemplar selection. The total cost is given first; the portion of this used by active selection is given next, followed by the number of exemplars selected, n, and finally the number of restarts used during training. Likely due to the number of restarts, the overhead of the ΔISB method in the first row was quite low, at 2.5%. Training upon actively selected exemplars required only 11.2% of the multiplies required by the baseline. The ten restarts gave active selection a clear advantage, as the bulk of the baseline cost was due to the high cost of the restarts when the network did not have sufficient complexity for the task.

To evaluate how much of the benefit of active selection is due to the number of restarts, we ran two more experiments in which the baseline was allowed to grow without any restarts if conjugate gradient fell into a local minima without fitting the exemplars within tolerance after a single run. This is not unreasonable, given the reliability of conjugate gradient, as we found that conjugate gradient would quite often fit a given candidate set on the first run, given a correctly specified network. However, occasionally conjugate gradient will fall into local minima, and so this approach is admittedly less reliable than using restarts. We grew the network trained by active selection using 5 restarts. This is the run illustrated in Fig. 6(i) and Fig. 7; the results are tabulated in the second row of Table III. On this particular run, conjugate gradient did select the correct size network using zero restarts, fitting the candidates on the first try with 3 hidden units. Active selection still required only 58.3% of the computation required by the baseline.

To evaluate the effect of having more candidate examples, we performed by experiment again using 500 candidates, again allowing the baseline method 0 restarts, while the active selection method used 5 restarts. However, in this case we had to run the baseline again to allow it to converge to the correct size network, as it was unable to fit the candidates on the first try when it grew to 3 hidden units. (We mark this result with an asterisk in Table III to remind that the cost of the previous run is not included.) This run gives a comparison of the typical cost of using active selection with the cost for the baseline when it does converge. The benefit of the ΔISB method was even greater than it was on the smaller set of candidates, requiring only 22.2% of the baseline cost to fit the 500 candidates. In this case, whereas the baseline failed to fit the candidates with smallest network on the first run, the ΔISB algorithm determined it correctly the first time.

To get a better comparison of average results, we ran 5 runs of each method upon a set of 200 candidates, and then repeated the comparison with 5 runs each upon a set of 500 candidates. Each of the four sets training runs was initialized using the same set of 5 initial randomized weights. For a conservative comparison we require the network trained by active selection determine the correct network complexity, but give the baseline method the advantage of being provided with the correct number of hidden units. The baseline method is restarted if it fails to fit the candidates. The cost of the restarts is summed into the total cost for each run. To eliminate any possible influence of our experience biasing the experiment through the initial choice of rmse_n, we set $\mathrm{tol}_n = \mathrm{tol}_N$ in all of these runs. In our experience this is not the best initial setting for rmse_n but it is reasonable. The results are tabulated in Tables IV and V. Table IV compares the cost of learning the skewed hump function using 200 candidate training examples. The "baseline" column gives the total cost for all 200 examples. The next set of columns pertain to active exemplar selection. The total cost is given along with the ratio of the total cost with the baseline cost in parentheses. The "overhead" gives the portion of the total cost due to exemplar selection. The number of exemplars selected is given by n. Cost is measured in millions of multiplies. The bottom row gives medians over the five runs. Baseline runs 1 and 2 required 1 and 2 restarts, respectively. All the other baseline runs fit the candidates within tolerance on the first attempt. For the comparison of the cost of learning the skewed hump function using 500 candidate training examples, the "baseline" column gives the total cost of training over all 500 examples. The next set of columns pertain to active exemplar selection. The total cost is given along with the ratio of the total cost with the baseline cost in parentheses. The "overhead" is the portion of the total cost due to exemplar selection. The number of exemplars selected is given by n. Cost is measured in measured in millions of multiplies. The bottom row gives medians over the five runs. Baseline runs

TABLE IV

Skewed hump, $N = 200$.

Run	Baseline cost	Active selection cost	Active selection overhead	n
1.	83.30	33.46 (40.2%)	2.60%	11
2.	381.43	31.46 (8.2%)	5.73%	16
3.	80.37	18.52 (23.0%)	6.49%	12
4.	61.08	78.28 (128.2%)	2.16%	15
5.	70.96	34.5 (48.6%)	4.81%	14
Med:	**80.37**	**33.46 (41.6%)**	**4.81%**	

TABLE V

Skewed hump, $N = 500$.

Run	Baseline cost	Active selection cost	Active selection overhead	n
1.	940.68	37.26 (4.0%)	8.56%	12
2.	182.67	33.17 (18.2%)	8.86%	11
3.	778.51	30.52 (3.9%)	17.20%	17
4.	1287.39	38.05 (3.0%)	12.51%	16
5.	241.18	44.43 (18.4%)	9.48%	15
Med:	**778.51**	**37.26 (4.8%)**	**9.48%**	

2 and 5 fit the candidates on the first attempt; baseline runs 1, 3, and 4 required 3,1, and 2 restarts, respectively. Note that active selection typically required much less computation overall, even though it is responsible for the additional task of determining the appropriate network complexity. Note further that while the cost of the baseline method for $N = 500$ was typically much greater than for $N = 200$, the cost of active selection was typically only slightly higher. (The average cost of active selection actually decreased.)

These runs (and in particular, Run 4 in Table IV) indicate how the choice of rmse_n can affect the cost of active selection. Observe that several of the runs selected more than 10 exemplars, the number that was found when $\mathrm{tol}_n = \mathrm{tol}_N/10 = 0.001$ (see Table III). However, the typical cost of these runs was still on par with the cost that resulted from using $\mathrm{tol}_n = 0.001$. The exception is Run 4 in Table IV, which cost more than the baseline. We repeated this particular run, reducing tol_n by half to 0.005. this improved the cost substantially, resulting in 32.24 million multiplies instead of 78.28 million, selecting 12 exemplars rather than 15.

VI. Discussion

We now turn to issues relevant to the practical application of our selection method. We emphasize the importance of obtaining a good solution to the training problem at each stage. We experimented briefly with an iterative training regime, in which little optimization was performed between successive examples. There was a tendency for the criterion to sample repetitively near previously sampled points when the optimization between successive examples was not stringent. In some such cases a pathological condition emerged in which the technique entered a cycle, endlessly alternating between two examples. For this reason, we advocate stringent optimization at each step.

We now address the overhead due to the selection process. The cost of selecting a new exemplar is about the same as that of making two iterations through the entire set of candidates with a steepest descent learning rule. This can be cut in half, at the expense of allocating space equal to the size of the set of candidates, by storing partial results and making one iteration through the set of candidates. However, we expect that this will be unreasonable for large data sets. When dealing with large data sets the cost of exemplar selection could be reduced by appropriately subsampling the set of candidates [32].

Observe that the selection cost is incurred only occasionally, because typically network training will process a given set of exemplars for numerous iterations before selecting a new exemplar. When multistarts are used to determine the appropriate network complexity for the task, several restarts might be performed before a new exemplar is selected. In our experience we have noticed that when multistarts are used the overhead of exemplar selection tends to be a small percentage of the total cost of training.

In our experience with the technique, we have noticed that the $\Delta\mathrm{ISB}$ criterion can have multiple maxima. It may be possible to exploit this to try to reduce the cost of obtaining exemplars further by selecting more than one exemplar at a time. Rather than just selecting the single exemplar maximizing our criterion, we might select a *set* of exemplars that locally maximize the criterion [22]. This reduces to the task of finding the modes of the selection criterion.

Arbitrary decisions were made in the derivation of the active selection method we obtained. We now discuss the effects of these choices, and how they might be modified to obtain variants of the method for practical use on particular applications. It is an open question as to how sensitive the method is to $H(x^n, w_n)$, the estimate of the Hessian appearing in (9). Observe that the appearance of H is due to our choice of weight update. This choice could be modified, depending upon the degree of approximation that is deemed necessary for the particular learning task. Our experiments demonstrate that the method can work well even when H is approximated. Recall that in our demonstrations we approximated or replaced H for computation simplicity. This was sufficient for the intended purpose of illustrating the method, although we advocate calculating H directly whenever feasible. That the method still selected concise training sets and saved computation indicates that even the crude approximation of H we used may be adequate on some tasks.

When obtaining the most concise set of exemplars is the highest priority, we expect the calculating H directly will be desirable. We may go even further, calculating the Hessian directly, taking appropriate steps to detect when it becomes ill-behaved. The numerical optimization literature offers considerable guidance in this regard. It is a straightforward modification to use the actual Hessian in place of H. This is equivalent to replacing the Gauss–Newton weight update in our derivation with the Newton update.

We employ the least squares performance criterion. It would be straightforward to replace it with another choice. For example, one might use cross entropy for use on classification tasks [38]. The estimator could be modified to include a (differentiable) complexity regularization term as well, such as weight decay [10]. In this case, exemplars would be selected according to their potential worth in minimizing the sum of network complexity and estimation error.

Finally, we summarize our initial experience with the technique on more difficult learning tasks. Space limitations prevent us from discussing all of the experiments we have done so far, but we have applied the technique to learning tasks requiring networks with multiple inputs and outputs with similarly favorable results. More extensive experiments are necessary to corroborate these preliminary results, especially on learning tasks requiring high dimensional data sets.

VII. Conclusion

The goal of this work is to explore the benefits of selective data handling. The ΔISB criterion we obtain is reasonable to compute, exhibits satisfying behavior, and is nonparametric in the sense that the complexity of the network need not be prespecified. Parameterization of the network function is necessary to evaluate training examples, but modifying the number of network parameters (weights) between successive training examples is allowed.

We demonstrate that the technique selects concise training sets, and that it works well in conjunction with a particular complexity regularization technique. The technique resulted in an overall computational savings for network training on the problems we tried. As might be expected, we found the cost benefit to depend upon the abundance of data. We expect the benefit to be amplified when used with a complexity regularization technique requiring a large number of training runs over a given data set, due to the low cost of training over a concise sets of exemplars.

The result we obtain has immediate practical applications. It may provide an advantage in domains where data is abundant, by pruning the training set to a manageable size, or with use in conjunction with data-driven network training algorithms. Our preliminary results indicate it may be useful in general purpose use as well, by reducing computation. More extensive benchmark tests are required on problems involving high-dimensional data sets.

References

[1] Y. S. Abu-Mostafa, "The Vapnik–Chervonenkis dimension," in *Neural Computation.* Cambridge, MA: MIT Press, 1989, no. 1, vol. 3, pp. 312–317.

[2] S. Ahmad and S. Omohundro, "A network for extracting the locations of point clusters using selective attention," Tech. Rep. 90–011, Int. Computer Science Institute, University of California, Berkeley; also to appear in *12th Ann. Conf. Cognitive Science Society.*

[3] A. Ash and A. Hedayat, "An introduction to design optimality with an overview of the literature," *Communications in Statistics, Part A—Theory and Methods*, vol. 7, pp. 1295–1325, 1978.

[4] R. G. Bartle, *The Elements of Integration.* New York, Wiley, 1966.

[5] E. B. Baum, "Neural net algorithms that learn in polynomial time from examples and queries," *IEEE Trans. Neural Networks*, vol. 2, pp. 5–19, 1991.

[6] E. B. Baum and D. Haussler, "What size network gives valid generalization," in *Neural Computation.* Cambridge, MA: MIT Press, 1989, vol. 1, no. 1, pp. 151–160.

[7] P. Billingsley, *Probability and Measure.* New York: Wiley, 1986.

[8] A. L. Rivest, "Training a 3-node neural network is NP-complete," in *Computational Learning Theory (COLT)*, Haussler (UCSC) and Pitt (Illinois), Eds. Cambridge, MA: Aug. 1988. (Sponsored by ACM SIGACT/SIGART, UC Santa Cruz, 1988.)

[9] G. Box and N. Draper, *Emirical Model-Building and Response Surfaces.* New York: Wiley, 1987.

[10] W. L. Buntine, and A. S. Weigend, "Bayesian back-propagation," to appear in *Complex Systems*, 1992.

[11] D. Cohn, L. Atlas, and R. Ladner, "Training connectionist networks with queries and selective sampling," in *Advances in Neural Information Processing Systems 2, Proc. Neural Information Processing Systems Conf.* San Mateo, CA: Morgan Kaufmann, 1990.

[12] D. Cohn, "A local approach to optimal queries," in *Connectionist Models Summer School (CMSS-90). Proc. 1990 Summuer School in San Diego*, D. S. Touretzky *et al.*, Eds. San Mateo, CA: Morgan Kaufmann.

[13] M. Cooray-Wijesinha, and A. I. Khuri, "The sequential generation of multiresponse D-optimal designs when the variance-covarance matrix is not known," *Comm. Statistics—Simulation*, vol. 16, no. 1, pp. 239–259, 1987.

[14] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electronic Computers*, vol. EC-14, pp. 326–334, 1965.

[15] J. J. Faraway, "Sequential design for the nonparametric regression of curves and surfaces," Tech. Rep. 177, Department of Statistics, The University of Michigan, Ann Arbor, 1990.

[16] D. Farmer, and J. Sidorowich, "Predicting chaotic time series," *Phys. Rev. Lett.*, vol. 59, pp. 845–848, 1987.

[17] V. V. Fedorov, *Theory of Optimal Experiments.* New York: Academic, 1972.

[18] A. Lapedes, and R. Farber, "How neural nets work," *Neural Information Processing Systems*, (Denver 1987), D. Z. Anderson, Ed. New York: American Institute of Physics, 1988, pp. 442–456.

[19] M. C. Mackey, and L. Glass. "Oscillation and chaos in physiological control systems," *Science*, vol. 197, p. 287, 1977.

[20] W. I. Gasarch, and M. B. Pleszkoch, "Learning via queries to an oracle," in *Proc. COLT'89 (Second Annual Workshop on Computational Learning Theory)*, 1989.

[21] L. Hellerstein and M. Karpinski, "Learning read-once formulas using membership queries," in *Proc. COLT'89 (Second Annual Workshop on Computational Learning Theory)*, 1989.

[22] G. Hinton, personal communication, 1991.

[23] K. Hornik, M. Stinchcombe, and H. White, "Multi-layer feedforward networks are universal approximators," in *Neural Networks.* New York: Pergamon, 1989, vol. 2, no. 5, pp. 359–366.

[24] J. N. Hwang, J. J. Choi, S. Oh, and R. J. Marks II, "Query-based learning applied to partially trained multi-layer perceptrons," *IEEE Trans. Neural Networks*, vol. 2, pp. 131–136. Jan. 1991.

[25] J. N. Hwang, J. J. Choi, S. Oh, and R. J. Marks II, "Query learning based on boundary search and gradient computation of trained multilayer perceptrons," in *Proc. IJCNN 1990, San Diego, The Int. Joint Conf. Neural Networks.*, 1990.

[26] S. Judd, "On the complexity of loading shallow neural networks," *J. Complexity*, vol. 4, pp. 177–192, 1988.

[27] S. Judd, "Learning in neural networks (extended abstract)," in *Computational Learning Theory (COLT)*, Haussler (UCSC) and Pitt (Illinois), Eds. Cambridge, MA: MIT Press, Aug. 1988. (Sponsored by ACM SIGACT/SIGART, UC Santa Cruz, 1989)

[28] A. I. Khuri, and J. A. Cornell, *Response Surfaces (Designs and Analyses).* New York: Marcel Dekker, Inc., 1987.

[29] S. A. Kurtz, and C. H. Smith, "On the role of search for learning," in *Proc. COLT'89 (Second Annual Workshop on Computational Learning Theory.*), 1989.

[30] D. J. C. MacKay, "Information-based objective functions for active data selection," submitted to *Neural Computation*, 1991.

[31] ——, "The evidence framework applied to classification networks," submitted to *Neural Computation*, 1991.

[32] M. Moller, "Supervised learning on large redundant training sets," to be published, 1991.

[33] H. G. Müller, "Optimal designs for nonparametric kernel regression," *Statistics and Probability Letters 2*, pp. 285–290, 1984.

[34] R. H. Myers, A. I. Khuri, and W. H. Carter, Jr., "Response surface methodology: 1966-1988," *Technometrics* vol. 31, no. 2, 1989.

[35] J. Platt "A resource-allocating neural network for function interpolation," *Neural Computation*, vol. 3, no. 2, pp. 213–225, 1991.

[36] M. E. Plutowski, and H. White, "Active selection of training examples for network learning in noiseless environments," Tech. Rep. CS91-180, Department of Computer Science and Engineering, The University of California, San Diego, La Jolla, CA, 1991.

[37] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C.* Cambridge, MA: Cambridge University Press, 1988.

[38] T. Sejnowski, personal communication, 1991.

[39] G. A. F. Seber, and C. J. Wild, *Nonlinear Regression.* New York: Wiley, 1989.

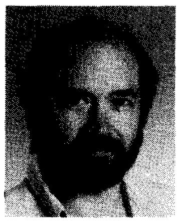
[40] L. G. Valiant, "A theory of the learnable," *Comm. ACM*, vol. 27, no. 11, 1984.

[41] H. White, "Learning in artificial neural networks: A statistical perspective," *Neural Computation.* Cambridge, MA: MIT Press, 1989, vol. 1, no. 4, pp. 425–464.

[42] H. White, "Connectionist nonparametric regression: Multilayer feedforward network can learn arbitrary mappings," in *Neural Networks.* New York: Pergamon, vol. 3, no. 5, pp. 535–549.

[43] S. Yakowitz, and E. Lugosi, "Random search in the presence of noise, with application to machine learning," *Society for Industrial and Applied Mathematics Journal on Scientific and Statistical Computing*, vol. 11, no. 4, pp. 702–712, 1990.

Mark Plutowski was born near Grand Forks, ND, where he received the B.S.E.E. degree from the University of North Dakota, Grand Forks. He received the M.S.E.E. degree from the University of Southern California in Los Angeles, and the Candidate of Philosophy degree from the Computer Science and Engineering department at the University of California, San Diego. He is currently working towards the Ph.D. degree at UCSD, with the objective of continuing neural network research and development.



Halbert White received the A. B. degree from Princeton University, NJ in 1972 and the Ph.D. from MIT in 1976.

He was elected Fellow of the Econometric Society in 1983, and was awarded a Guggenheim fellowship in 1988 for the study of artificial neural networks.

He is Professor of Economics at the University of California, San Diego. His current research interests are in mathematical foundations and applications of artificial neural networks, and in econometric theory and applications.