Brief Papers

A General Regression Neural Network

Donald F. Specht

Abstract—**This paper describes a memory-based network that provides estimates of continuous variables and converges to the underlying (linear or nonlinear) regression surface. This general regression neural network (GRNN) is a one-pass learning algorithm with a highly parallel structure. Even with sparse data in a multidimensional measurement space, the algorithm provides smooth transitions from one observed value to another. The algorithmic form can be used for any regression problem in which an assumption of linearity is not justified. The parallel network form should find use in applications such as learning the dynamics of a plant model for prediction or control.**

I. Introduction

EXTENSIVE effort has been devoted to developing techniques for identification of linear time-invariant systems (for example, see [1] and [2]). The linear identification is based on measured input and output values of the system. Identification for nonlinear systems is also based on measured input and output values, but it is more difficult. Recently Narendra and Parthasarathy [3] have presented a procedure that uses neural networks for identification and control of nonlinear systems. For identification, the input and output values of the system are fed into a multilayer neural network. Although that paper used the back-propagation algorithm [4] for training the networks, the same identification and control framework can be used with neural networks having other characteristics.

One disadvantage of back-propagation is that it can take a large number of iterations to converge to the desired solution. An alternative to back-propagation that has been used in classification is the probabilistic neural network (PNN) [5], [6], which involves one-pass learning and can be implemented directly in neural network architecture. This paper describes a similar one-pass neural network learning algorithm which can be used for estimation of continuous variables.

If the variables to be estimated are future values, then the procedure is a *predictor*. If the variable or variables to be estimated relate output variables to input variables, then the procedure can be used to *model* the process or system. Once the system has been modeled, a control function can be defined. If the procedure is taught samples of a control function, it can estimate the entire control function, and it becomes a controller.

Manuscript received October 26, 1990; revised May 3, 1991. This work was supported by the Lockheed Missiles and Space Company, Inc., Independent Research Project RDD360 (Neural Network Technology).

The author is with the Lockheed Palo Alto Research Laboratory, 3251 Hanover Street, Palo Alto, CA 94304.

IEEE Log Number 9101552.

Subsection II-A of this paper provides the mathematical derivation of a procedure. Subsection II-B describes the normalization required for the input data and a method for selecting the proper value of the smoothing parameter. Subsection II-C extends the method to employ clustering and to adapt to changing conditions. In subsection II-D the method is compared with similar techniques, such as the procedures of Moody and Darken [7], radial basis functions [8], [9], the cerebellar model arithmetic computer (CMAC) [10], and classical regression techniques. Hardware implementation is discussed in subsection II-E. Section III presents two examples. The first example models a simple nonlinear gain. The second example, with five inputs and one output, was first described in [3] and solved using a back-propagation neural network (BPN). For the second example this technique required only 1% of the training samples required by the back-propagation model to achieve comparable accuracies.

II. Approach

The regression of a dependent variable, Y, on an independent variable, X, is the computation of the most probable value of Y for each value of X based on a finite number of possibly noisy measurements of X and the associated values of Y. The variables X and Y are usually vectors. In system identification, the dependent variable, Y, is the system output and the independent variable, X, is the system input. In order to implement system identification, it is usually necessary to assume some functional form with unknown parameters a_i. The values of the parameters are chosen to make the best fit to the observed data. In the case of linear regression, for example, the output, Y, is assumed to be a linear function of the input, X, and the unknown parameters, a_i, are linear coefficients. The approach presented here uses a method that frees it from the necessity of assuming a specific functional form. Rather, it allows the appropriate form to be expressed as a probability density function (pdf) that is empirically determined from the observed data using Parzen window estimation [11]. Thus, the approach is not limited to any particular form and requires no prior knowledge of the appropriate form.

If one knows the joint pdf of x and y, the conditional pdf and the expected value can be computed. In this paper, the joint pdf will be estimated from examples using nonparametric estimators. The resulting regression equation can be implemented in a parallel, neural-network-like structure. Since the parameters of the structure are deter-

1045-9227/91$01.00

mined directly from examples rather than iteratively, the structure "learns" and can begin to generalize immediately. To the extent that the network is implemented in parallel hardware, it also can estimate values of Y for any new value of $\boldsymbol{X}$ in the short time determined by the propagation time through four layers of a neural network.

A. General Regression

Assume that $f(\boldsymbol{x}, y)$ represents the known joint continuous probability density function of a vector random variable, $\boldsymbol{x}$, and a scalar random variable, y. Let $\boldsymbol{X}$ be a particular measured value of the random variable $\boldsymbol{x}$. The conditional mean of y given $\boldsymbol{X}$ (also called the regression of y on $\boldsymbol{X}$) is given by

$$E[y|\boldsymbol{X}] = \frac{\int_{-\infty}^{\infty} y f(\boldsymbol{X}, y)\, dy}{\int_{-\infty}^{\infty} f(\boldsymbol{X}, y)\, dy}. \tag{1}$$

When the density $f(\boldsymbol{x}, y)$ is not known, it must usually be estimated from a sample of observations of $\boldsymbol{x}$ and y. For a nonparametric estimate of $f(\boldsymbol{x}, y)$, we will use the class of consistent estimators proposed by Parzen [11] and shown to be applicable to the multidimensional case by Cacoullos [12]. As noted in [13], these estimators are a good choice for estimating the probability density function, f, if it can be assumed that the underlying density is continuous and that the first partial derivatives of the function evaluated at any $\boldsymbol{x}$ are small. The probability estimator $\hat{f}(\boldsymbol{X}, Y)$ is based upon sample values $\boldsymbol{X}^i$ and Y^i of the random variables $\boldsymbol{x}$ and y, where n is the number of sample observations and p is the dimension of the vector variable $\boldsymbol{x}$:

$$\hat{f}(\boldsymbol{X}, Y) = \frac{1}{(2\pi)^{(p+1)/2}\sigma^{(p+1)}} \cdot \frac{1}{n}\sum_{i=1}^{n} \exp\left[-\frac{(\boldsymbol{X}-\boldsymbol{X}^i)^T(\boldsymbol{X}-\boldsymbol{X}^i)}{2\sigma^2}\right] \cdot \exp\left[-\frac{(Y-Y^i)^2}{2\sigma^2}\right]. \tag{2}$$

A physical interpretation of the probability estimate $\hat{f}(\boldsymbol{X}, Y)$ is that it assigns sample probability of width σ for each sample $\boldsymbol{X}^i$ and Y^i, and the probability estimate is the sum of those sample probabilities. Substituting the joint probability estimate $\hat{f}$ in (2) into the conditional mean, (1), gives the desired conditional mean of y given $\boldsymbol{X}$. In particular, combining (1) and (2) and interchanging the order of integration and summation yields the desired conditional mean, designated $\hat{Y}(\boldsymbol{X})$:

$$\hat{Y}(\boldsymbol{X}) = \frac{\sum_{i=1}^{n} \exp\left[-\frac{(\boldsymbol{X}-\boldsymbol{X}^i)^T(\boldsymbol{X}-\boldsymbol{X}^i)}{2\sigma^2}\right] \int_{-\infty}^{\infty} y \exp\left[-\frac{(y-Y^i)^2}{2\sigma^2}\right] dy}{\sum_{i=1}^{n} \exp\left[-\frac{(\boldsymbol{X}-\boldsymbol{X}^i)^T(\boldsymbol{X}-\boldsymbol{X}^i)}{2\sigma^2}\right] \int_{-\infty}^{\infty} \exp\left[-\frac{(y-Y^i)^2}{2\sigma^2}\right] dy} \tag{3}$$

Defining the scalar function D_i^2,

$$D_i^2 = (\boldsymbol{X}-\boldsymbol{X}^i)^T(\boldsymbol{X}-\boldsymbol{X}^i) \tag{4}$$

and performing the indicated integrations yields the following:

$$\hat{Y}(\boldsymbol{X}) = \frac{\sum_{i=1}^{n} Y^i \exp\left(-\frac{D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^{n} \exp\left(-\frac{D_i^2}{2\sigma^2}\right)}. \tag{5}$$

Because the particular estimator, (3), is readily decomposed into x and y factors, the integrations were accomplished analytically. The resulting regression, (5), which involves summations over the observations, is directly applicable to problems involving numerical data.

Parzen [11] and Cacoullos [12] have shown that density estimators of the form of (2) used in estimating (1) by (5) are consistent estimators (asymptotically converging to the underlying probability density function $f(x, y)$) at all points (x, y) at which the density function is continuous, provided that $\sigma = \sigma(n)$ is chosen as a decreasing function of n such that

$$\lim_{n\to\infty} \sigma(n) = 0$$

and

$$\lim_{n\to\infty} n\sigma^p(n) = \infty. \tag{6}$$

The estimate $\hat{Y}(\boldsymbol{X})$ can be visualized as a weighted average of all of the observed values, Y_i, where each observed value is weighted exponentially according to its Euclidean distance from $\boldsymbol{X}$. When the smoothing parameter σ is made large, the estimated density is forced to be smooth and in the limit becomes a multivariate Gaussian with covariance $\sigma^2 I$. On the other hand, a smaller value of σ allows the estimated density to assume non-Gaussian shapes, but with the hazard that wild points may have too great an effect on the estimate. As σ becomes very large, $\hat{Y}(\boldsymbol{X})$ assumes the value of the sample mean of the observed Y^i, and as σ goes to 0, $\hat{Y}(\boldsymbol{X})$ assumes the value of the Y^i associated with the observation closest to $\boldsymbol{X}$. For intermediate values of σ, all values of Y^i are taken into account, but those corresponding to points closer to $\boldsymbol{X}$ are given heavier weight.

When the underlying parent distribution is not known, it is not possible to compute an optimum σ for a given number of observations n. It is therefore necessary to find σ on an empirical basis. This can be done easily when the density estimate is being used in a regression equation

because there is a natural criterion that can be used for evaluating each value of σ, namely, the mean squared error between Y^j and the estimate $\hat{Y}(X^j)$. For this purpose, the estimate in (5) must be modified so that the jth element in the summation is eliminated. Thus each $\hat{Y}(X^j)$ is based on inference from all the observations except the actual observed value at X^j. This procedure is used to avoid an artificial minimum error as $\sigma \to 0$ that results when the estimated density is allowed to fit the observed data points. Overfitting of the data is also present in the least-squares estimation of linear regression surfaces, but there it is not as severe because the linear regression equation has only $p + 1$ degrees of freedom. If $n >> p$, the phenomenon of overfitting is commonly ignored.

Y and $\hat{Y}$ can be vector variables instead of scalars. In this case, each component of the vector $\boldsymbol{Y}$ would be estimated in the same way and from the same observations $(\boldsymbol{X}, \boldsymbol{Y})$ except that Y is now augmented by observations of each component. It will be noted from (5) that the denominator of the estimator and all of the exponential terms remain unchanged for vector estimation.

The explanation here is similar to that in [14]. That paper goes on to approximate $f(x, y)$ with a truncated polynomial [15]. The estimator of (5) then becomes the ratio of two polynomials. Very recently, Grabec [16] used the same estimator to predict chaotic behavior. Based on application of the maximum entropy principle, he developed an iterative technique for optimal estimation with a reduced set of k exemplars, where $k < n$.

B. Normalization of Input and Selection of Value of Smoothing Parameter

As a preprocessing step, it is usually necessary to scale all input variables such that they have approximately the same ranges or variances. The need for this stems from the fact that the underlying probability density function is to be estimated with a kernel that has the same width in each dimension. This step is not necessary in the limit as $n \to \infty$ and $\sigma \to 0$, but it is very helpful for finite data sets. Exact scaling is not necessary, so the scaling variables need not be changed every time new data are added to the data set.

It has been suggested that an even better estimation of densities using Parzen windows and finite data sets can be obtained using a different set of σ's for each exemplar [17]. This concept could be carried into the general regression equations too, but would negate the instant learning capability of the network as proposed.

After rough scaling has been accomplished, it is necessary to select the width of the estimating kernel, σ. A useful method of selecting σ is the *holdout method*. For a particular value of σ, the holdout method consists in removing one sample at a time and constructing a network based on all of the other samples. Then the network is used to estimate Y for the removed sample. By repeating this process for each sample and storing each estimate, the mean-squared error can be measured between the actual sample values Y^i and the estimates. The value of σ giving the smallest error should be used in the final network. Typically, the curve of mean-squared error versus σ exhibits a wide range of values near the minimum, so it is not difficult to pick a good value for σ without a large number of trials.

Finally, it should be pointed out that the Gaussian kernel used in (2) could be replaced by any of the Parzen windows. Several of these have been enumerated in [6] and expressed in terms of neural network nodes having specific activation functions. Particularly attractive from the point of view of computational simplicity is [6, eq. (21)], which results in the estimator

$$\hat{Y}(X) = \frac{\sum_{i=1}^{n} Y^i \exp\left(-\frac{C_i}{\sigma}\right)}{\sum_{i=1}^{n} \exp\left(-\frac{C_i}{\sigma}\right)} \tag{7}$$

where

$$C_i = \sum_{j=1}^{p} |X_j - X_j^i|. \tag{8}$$

This measure is often called the city block distance. In experience with PNN classification we have found that the city block metric works approximately as well as the Euclidian distance metric. It is assumed that this observation will hold true when using the same pdf estimators for nonlinear regression.

C. Clustering and Adaptation to Nonstationary Statistics

For some problems, the number of observations $(\boldsymbol{X}, Y)$ may be small enough that it is desired to use all the data obtainable directly in the estimator of (5) or (7). In other problems, the number of observations obtained can become sufficiently large that it is no longer practical to assign a separate node (or neuron) to each sample. Various clustering techniques can be used to group samples so that the group can be represented by only one node that measures distance of input vectors from the cluster center. Burrascano [18] has suggested using learning vector quantization to find representative samples to use for PNN to reduce the size of the training set. This same technique also can be used for the current procedure. Likewise, K-means averaging [19], adaptive K-means [7], one-pass K-means clustering, or the clustering technique used by Reilly *et al.* [19] for the restricted Coulomb energy (RCE) network could be used. However the cluster centers are determined, let us assign a new variable, N_i, to indicate the number of samples that are represented by the ith cluster center. Equation (5) can then be rewritten as

$$\hat{Y}(\boldsymbol{X}) = \frac{\sum_{i=1}^{m} A^i \exp\left(-\frac{D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^{m} B_i \exp\left(-\frac{D_i^2}{2\sigma^2}\right)} \tag{9}$$

$$\left.\begin{aligned} A^i(k) &= A^i(k-1) + Y^j \\ B^i(k) &= B^i(k-1) + 1 \end{aligned}\right\} \text{ incremented each time a training observation } Y^j \text{ for cluster } i \text{ is encountered} \tag{10}$$

where $m < n$ is the number of clusters, and $A^i(k)$ and $B^i(k)$ are the values of the coefficients for cluster i after k observations. $A^i(k)$ is the sum of the Y values and $B^i(k)$ is the number of samples assigned to cluster i.

The method of clustering can be as simple as establishing a single radius of influence, r. Starting with the first sample point (X, Y), establish a cluster center, X^i, at X. All future samples for which the distance $|X - X^i|$ is less than the distance to any other cluster center and is also $\leq r$ would update equations (10) for this cluster. A sample for which the distance to the nearest cluster is $> r$ would become the center for a new cluster. Note that the numerator and denominator coefficients are completely determined in one pass through the data. No iteration is required to improve the coefficients.

Since the A and B coefficients can be determined using recursion equations, it is easy to add a forgetting function. This is desirable if the network is being used to model a system with changing characteristics. If equations (10) are written in the form

$$\left.\begin{aligned} A^i(k) &= \frac{\tau - 1}{\tau} A^i(k-1) + \frac{1}{\tau} Y^j \\ B^i(k) &= \frac{\tau - 1}{\tau} B^i(k-1) + \frac{1}{\tau} \end{aligned}\right\} \text{ new sample assigned to cluster } i$$

$$\left.\begin{aligned} A^i(k) &= \frac{\tau - 1}{\tau} A^i(k-1) \\ B^i(k) &= \frac{\tau - 1}{\tau} B^i(k-1) \end{aligned}\right\} \text{ new sample assigned to a cluster } \neq i \tag{11}$$

then τ can be considered the time constant of an exponential decay function (where τ is measured in update samples rather than in units of time). It is interesting to note that if all of the coefficients were attenuated by the factor $(\tau - 1)/\tau$, the regression equation (9) would be unchanged; however, the new sample information will have an influence in the local area around its assigned cluster center.

For practical considerations, there should be a lower threshold established for B^i, so that when sufficient time has elapsed without update for a particular cluster, that cluster (and its associated A^i and B^i coefficients) would be eliminated. In the case of dedicated neural network hardware, these elements could be reassigned to a new cluster.

When the regression function of (9) is used to represent a system that has many modes of operation, it is undesirable to forget data associated with modes other than the current one. To be selective about forgetting, one might assign a second radius, $\rho \gg r$. In this case, equations (11) would be applied only to cluster centers within a distance ρ of the new training sample.

Equation (12), the equivalent estimator using city block distances, can use the same coefficient update equations, (10) and (11).

$$\hat{Y}(X) = \frac{\sum_{i=1}^{m} A^i \exp\left(-\frac{C_i}{\sigma}\right)}{\sum_{i=1}^{m} B^i \exp\left(-\frac{C_i}{\sigma}\right)}. \tag{12}$$

Higher moments can also be estimated with y^q substituted for y in (1). Therefore variance of the estimate and standard deviation can also be estimated directly from the training examples.

D. Comparison with Other Techniques

Conventional nonlinear regression techniques involve either *a priori* specification of the form of the regression equation with subsequent statistical determination of some undetermined constants, or statistical determination of the constants in a general regression equation, usually of polynomial form. The first technique requires that the form of the regression equation be known *a priori* or guessed. The advantages of this approach are that it usually reduces the problem to estimation of a small number of undetermined constants, and that the values of these constants when found may provide some insight to the investigator. The disadvantage is that the regression is constrained to yield a "best fit" for the specified form of equation. If the specified form is a poor guess and not appropriate to the data base to which it is applied, this constraint can be serious. Classical polynomial regression is usually limited to polynomials in one independent variable or low order, because high-order polynomials involving multiple variates often have too many free constants to be determined using a fixed number, n, of observations (X^i, Y^i). A classical polynomial regression surface may fit the n observed points very closely, but unless n is much larger than the number of coefficients in the polynomial, there is no assurance that the error for a new point taken randomly from the distribution $f(x, y)$ will be small.

With the regression defined by (5) or (7), however, it is possible to let σ be small, which allows high-order curves if they are necessary to fit the data. Even in the limit as σ approaches 0, (5) is well behaved. It estimates $\hat{Y}(X)$ as being the same as the Y^i associated with the X^i that is closest in Euclidean distance to X (nearest neighbor estimator). For any $\sigma > 0$, there is a smooth interpolation between the observed points (as distinct from the discontinuous change of $\hat{Y}$ from one value to another at points equidistant from the observed points when $\sigma = 0$). Other methods used for estimating general regression surfaces include the back-propagation of errors neural network (BPN), radial basis functions (RBF's) [8], the method of Moody and Darken [7], CMAC [10], and the use of Ko-

honen mapping to find the knots for a multidimensional spline fit [22].

The principal advantages of GRNN are fast learning and convergence to the optimal regression surface as the number of samples becomes very large. GRNN is particularly advantageous with sparse data in a real-time environment, because the regression surface is instantly defined everywhere, even with just one sample. The one-sample estimate is that $\hat{Y}$ will be the same as the one observed value regardless of the input vector X. A second sample will divide hyperspace into high and low halves with a smooth transition between them. The surface becomes gradually more complex with the addition of each new sample point.

The principal disadvantage of the technique of (5) is the amount of computation required of the trained system to estimate a new output vector. The version of (9)–(11) using clustering overcomes this problem to a large degree. Soon the development of neural-network semiconductor chips capable of performing all the indicated operations in parallel will greatly speed performance. Almost all the neurons are pattern units, and are identical. The step and repeat microlithography techniques of semiconductor manufacturing are ideally suited to replicating large numbers of identical cells.

The algorithms of (5) or (7) can be used for rapid development of applications because they are easy to use. When further adaptation is not required, an estimation technique that requires less computation for evaluation of new samples could be substituted. For this purpose, the following techniques could be used: (1) a least-squares fit of a general polynomial, (2) a feedforward neural network based on the back-propagation of errors paradigm, or (3) use of learning vector quantization or other clustering methods to find a reduced set of exemplars.

The general polynomial is essentially not usable unless a very large number of exemplars are available. However, (5) or (7) can be used to define the regression surface everywhere and then can be used to "manufacture" a sufficient number of exemplars. Back-propagation (BPN) requires very long training times [23] and is subject to converging to local minima instead of finding the global minimum error surface. For a mature application, the long training time may be justified. To guard against getting trapped in local minima, the BPN can be compared with the GRNN accuracy. If the BPN accuracy is not comparable, the BPN adaptation should be restarted with different initial conditions until comparable accuracy is obtained.

For interpolation applications, the procedure averages the values of sample points using the Parzen window function. When the samples are subject to measurement errors or stochastic variations, then averaging of this type is desirable. However, when the given sample values are presumed to be accurate and it is desired to perform a smooth interpolation between them, then the averaging of many samples will result in errors at the location of the sample points. A preferable solution in this case is the method of radial basis functions, in which a linear combination of exponential terms is found that gives no error at the sample points. Generally, RBF neurons are not identical and require global computations to determine their parameters.

The technique most similar to the proposed estimator is that proposed by Moody and Darken [7]. The structure of the network that implements the ratio of the sums of exponentials is, indeed, identical for one variant of each method. However, the training methods are quite different. Whereas Moody and Darken (hereafter referred to as M&D) use least mean squared adaptation to find the coefficients in (5) or (7), the proposed method uses observed samples (or averages of observed samples) directly as these coefficients. This may not seem like much of a difference, but it results in three advantages:

1) The estimate of (5) or (7) is the weighted average of actual sample points. This fact establishes that the estimate is bounded to the range of the observed samples.
2) The variable σ is a smoothing parameter that can be made large to smooth out noisy data or small to allow the estimated regression surface to be as nonlinear as is required to approximate closely the actual observed values of Y^i.
3) The coefficients are determined in one pass through the data; no iterative algorithm is required.

In contrast, there is no constraint on the M&D algorithm to estimate values within the range of the observed samples. A combination of large σ and sparse, noisy samples can produce artificial minima and maxima well outside the range of observed samples, Y^i. At the limiting condition of $\sigma = 0$, both algorithms converge to become the nearest neighbor estimator. For larger numbers of observations, the averaging inherent in the M&D procedure reduces its sensitivity to noise.

CMAC [10] represents another similar structure for estimating continuous output variables based on stored experience. In this structure the neurons closest (in Euclidian distance) to the input vector turn on and the others produce zero output. These neurons are less complicated than those of RBF, M&D, or the present method, but more are required to produce a smoothly varying output function.

E. Neural-Network Implementation

An artificial neural network is usually defined as a network composed of a large number of simple processors (neurons) that are massively interconnected, operate in parallel, and learn from experience (examples). These are the primary known characteristics of biological neural systems that are the easiest to exploit in artificial neural systems. Fig. 1 shows the overall block diagram of the present proposal in its adaptive form represented by (9) or (12). The input units are merely distribution units, which provide all of the (scaled) measurement variables

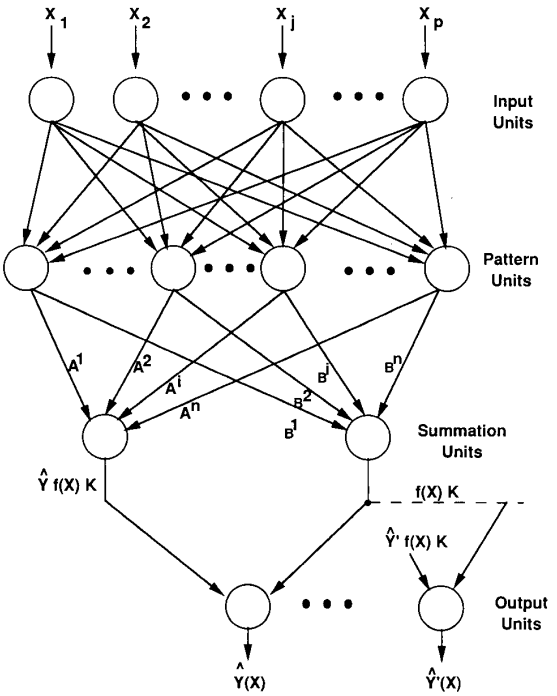

Fig. 1. GRNN block diagram.

X to all of the neurons on the second layer, the pattern units. The pattern unit is dedicated to one exemplar or one cluster center. When a new vector X is entered into the network, it is subtracted from the stored vector representing each cluster center. Either the squares or the absolute values of the differences are summed and fed into a nonlinear activation function. The activation function normally used is the exponential, although all of the alternate activation functions shown in [6, table 1] also could be used here. The pattern unit outputs are passed on to the summation units.

The summation units perform a dot product between a weight vector and a vector composed of the signals from the pattern units. The summation unit that generates an estimate of $f(X)K$ sums the outputs of the pattern units weighted by the number of observations each cluster center represents. When using (11), this number is also weighted by the age of the observations. K is a constant determined by the Parzen window used, but is not data-dependent and does not need to be computed. The summation unit that estimates $\hat{Y}f(X)K$ multiplies each value from a pattern unit by the sum of samples Y^j associated with cluster center X^i. The output unit merely divides $\hat{Y}f(X)K$ by $f(X)K$ to yield the desired estimate of Y.

When estimation of a vector Y is desired, each component is estimated using one extra summation unit, which uses as its multipliers sums of samples of the component of Y associated with each cluster center X^i. There may be many pattern units (one for each exemplar or cluster center); however, the addition of one element in the output vector requires only one summation neuron and one output neuron.

What is shown in Fig. 1 is a feedforward network that can be used to estimate a vector Y from a measurement vector X. Because they are not interactive, all of the neurons can operate in parallel. Not shown in Fig. 1 is a microprocessor that assigns training patterns to cluster centers and updates the coefficients A^i and B^i. One practical advantage of not migrating the cluster centers, depending on the hardware implementation chosen, is that the cluster center data could be written into the pattern units using a programmable read-only memory. Then the only weights that have to be adaptive and readily changed are the A^i and B^i weights in the summation layer. Alternatively, the cluster centers could also be adjusted using K-means averaging [19].

III. Examples

Estimators of the type described have many potential uses as models, inverse models, and controllers. Two examples are presented here. The first is a contrived example to illustrate the behavior of the estimated regression line. The second is an identification problem in controls first posed by Narendra and Parthasarathy [3].

A. A One-Dimensional Example

A simple problem with one independent variable will serve to illustrate some of the differences between the techniques that have been discussed. Suppose that a regression technique is needed to model a "plant" which happens to be an amplifier that saturates in both polarities and has an unknown offset. Its input/output (I/O) characteristic is shown in Fig. 2. With enough sample points, many techniques would model the plant well. However, in a large dimensional measurement space, any practical data set appears to be sparse. The following illustration shows how the methods work on this example with sparse data, namely, five samples at $x = -2, -1, 0, 1$, and 2. When polynomial regression using polynomials of first, second, and fourth order was tried, the results were predictable. The polynomial curves are poor approximations to the "plant" except at the sample points. In contrast, Fig. 3 shows the I/O characteristic of this same plant as estimated by GRNN. Since GRNN always estimates using a (nonlinearly) weighted average of the given samples, the estimate is always within the observed range of the dependent variable. In the range from $x = -4$ to $x = 4$, the estimator takes on a family of curves depending on σ. Any curve in the family is a reasonable approximation to the plant of Fig. 2. The curve corresponding to $\sigma = 0.5$ is the best approximation. Larger values of σ provide more smoothing, and lower values provide a closer approximation to the sample values plus a "dwell" region at each sample point. When the holdout method was used to select σ, $\sigma = 0.3$ was selected (based on only four sample points at a time).

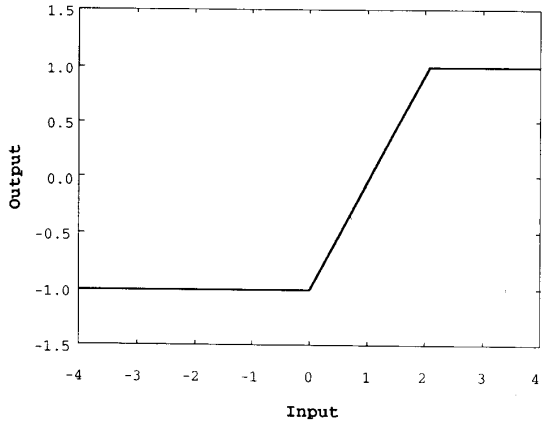

Fig. 2. Input/output characteristics of simple "plant."

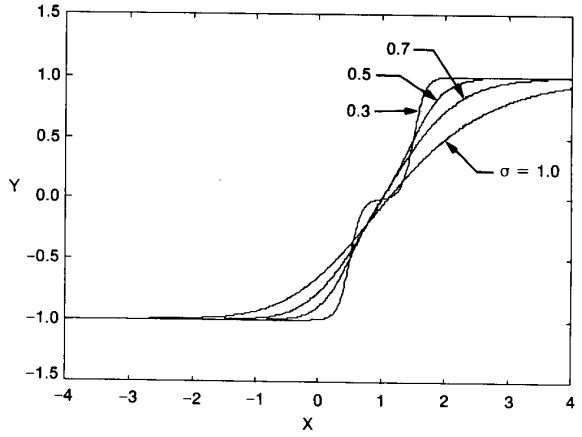

Fig. 3. Input/output characteristics of plant of Fig. 2 as estimated by GRNN based on sample points at $X = -2, -1, 0, 1$, and 2.

The curve that would result from back-propagation would depend on a number of choices having to do with the configuration of hidden units, initial conditions, and other parameters. The main difference between Fig. 3 and the curve resulting from radial basis functions is that the RBF ordinate would decrease to zero for large values of $|X|$.

B. Adaptive Control Systems

The fields of nonlinear control systems and robotics are particularly good application areas that can use the potential speed of neural networks implemented in parallel hardware, the adaptability of instant learning, and the flexibility of a completely nonlinear formulation. A straightforward technique can be used. First, model the plant as in Fig. 4. The GRNN learns the relationships between the input vector (the input state of the system and the control variables) and the simulated or actual output of the system. Control inputs can be supplied by a nominal controller (with random variations added to explore

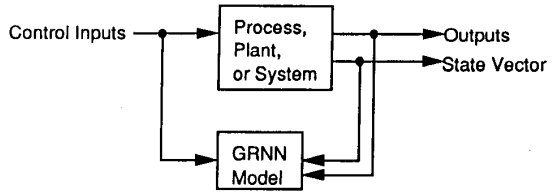

Fig. 4. Modeling the system.

inputs not allowed by the nominal controller) or by a human operator. After the model is trained, it can then be used to determine control inputs using an automated "what if" strategy or by finding an inverse model. Modeling involves discovering the association between inputs and outputs, so an inverse model can be determined from the same data base as the forward model by assigning the input variable(s) to the function of the desired output, Y, in Fig. 1, and the state vector and other measurements are considered components of the X vector in Fig. 1. One way in which the neural network could be used to control a plant is illustrated in Fig. 5.

Adaptive inverse neural networks can be used for control purposes either in the feedforward path or in the feedback path. Atkeson *et al.* [24] have used an adaptive inverse in the feedforward path with positional and velocity feedback to correct for residual error in the model. They noted that the feedback had less effect as the inverse model improved from experience. Atkeson *et al.* [24] have used a content addressable memory as the inverse model and have reported good results. Interestingly, the success reported was based on using only single nearest neighbors as estimators. Their paper mentions the possibility of extending the work to local averaging.

Farrell *et al.* [25] have used both sigmoidal processing units and Gaussian processing units as a neural network controller in a model reference adaptive control system. They note that the Gaussian processing units have an advantage in control systems because the localized influence of each Gaussian node allows the learning system to refine its control function in one region of measurement space without degrading its approximation in distant regions. The same advantage would hold true when using (9) or (12) as an adaptive inverse.

Narendra and Parthasarathy [3] separate the problem of control of nonlinear dynamical systems into an identification or system modeling section, and a model reference adaptive control (MRAC) section. Four different representations of the plant are described: 1) output of plant is a linear combination of delayed outputs plus a nonlinear combination of delayed inputs; 2) output of plant is a nonlinear combination of delayed outputs plus a linear combination of delayed inputs; 3) output of plant is a nonlinear combination of outputs plus a nonlinear combination of inputs; and 4) output of plant is a nonlinear function of delayed outputs and delayed inputs. GRNN, together with tapped delay lines for the outputs and inputs, could directly implement the identification task for the most gen-

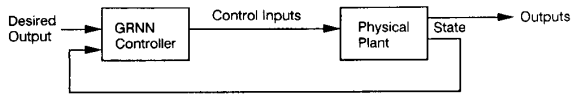

Fig. 5. A GRNN controller.

eral of these models (which subsumes the others). This is true both for single-input, single-output (SISO) plants and for multi-input, multi-output (MIMO) plants. Once the plant has been identified (modeled), all of the methods of [3], which are based on the back-propagation network (BPN) [4], can be used for adapting a controller to minimize the difference between the output of the reference model and the output of the identification model of the plant. This combination of technologies yields a controller with the simpler structure of BPN but still utilizes the instant learning capability of GRNN for the plant modeling function.

One of the more difficult examples given in [3] is example number 4, for which the authors identified the plant using the type 4 model above. In this example, the plant is assumed to be of the form

$$y_p(k+1) = f[y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)] \tag{13}$$

where $y_p(k+1)$ is the next time sample of the output of the plant, $y_p(k)$ is the current output, $y_p(k-1)$ and $y_p(k-2)$ are delayed time samples of the output, $u(k)$ is the current input, $u(k-1)$ is the previous input, and the unknown function f has the form

$$f[x_1, x_2, x_3, x_4, x_5] = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_2^2 + x_3^2}. \tag{14}$$

In the identification model, a GRNN network was used to approximate the function f. Fig. 6 shows the outputs of the plant and the model when the identification procedure was carried out for 1000 time steps using a random input signal uniformly distributed in the interval $[-1, 1]$, and using a $\sigma = 0.315$. In Fig. 6, the input to the plant and the identified model is given by $u(k) = \sin(2\pi k/250)$ for $k < 500$ and $u(k) = 0.8 \sin(2\pi k/250) + 0.2 \sin(2\pi k/25)$ for $k > 500$. Fig. 6 shows approximately the same amount of error between the model and the plant as does [3, fig. 16]; however, only 1000 time steps were required for the GRNN model to achieve this degree of accuracy, compared with 100 000 time steps required for the back-propagation model used in [3]. In other words, only 1% of the training was required to achieve comparable accuracies. The identification was accomplished with 1000 nodes in a five-dimensional space. No attempt has been made to reduce the number of nodes through clustering.

For back-propagation it may be that for the 100 000 presentations of training points, the same 1000 patterns could have been repeated 100 times each to achieve the same results. In this case it could be said that GRNN learned in one pass through the data instead of 100 passes.

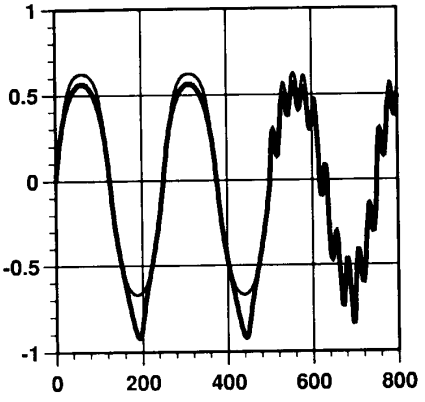

Fig. 6. Outputs of plant (dark line) and of the GRNN model (lighter line) after training with 1000 random patterns.

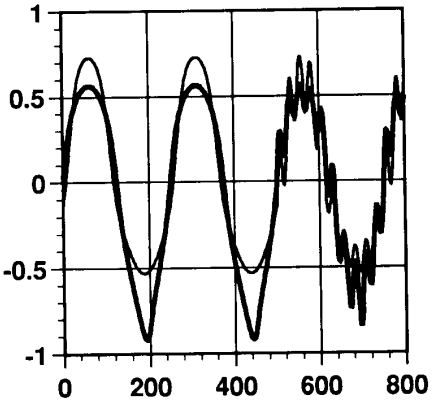

Fig. 7. Outputs of plant (dark line) and of the GRNN model (lighter line) after training with only ten random patterns.

An experiment was performed to determine the extent to which performance degrades with less training data. First the training set was reduced to 100 patterns, then to ten patterns. Fig. 7 shows the output of the plant and the model when the identification procedure was carried out for only the first ten random patterns of the 1000 (instead of the 100 000 used for BPN). The model predicted the plant output with approximately twice the error of the fully trained BPN, but using only 0.01% of the training data. Although it is not to be expected that equivalent performance would result from training with any ten random patterns, this performance was achieved on the first trial of only ten patterns. Certainly further research is indicated to provide estimates of the number of patterns required for the identification of important classes of systems.

IV. Summary and Conclusions

This paper describes a new network that provides estimates of continuous variables and converges to the underlying (linear or nonlinear) regression surface. The network features fast learning that does not require an

iterative procedure, and a highly parallel structure. It can be used for prediction, modeling, mapping, and interpolating or as a controller.

The general regression neural network (GRNN) is similar in form to the probabilistic neural network (PNN). Whereas PNN finds decision boundaries between categories of patterns, GRNN estimates values for continuous dependent variables. Both do so through the use of nonparametric estimators of probability density functions.

The advantages of GRNN relative to other nonlinear regression techniques are as follows.

1) The network "learns" in one pass through the data and can generalize from examples as soon as they are stored.
2) The estimate converges to the conditional mean regression surfaces as more and more examples are observed; yet, as indicated in the examples, it forms very reasonable regression surfaces based on only a few samples.
3) The estimate is bounded by the minimum and maximum of the observations.
4) The estimate cannot converge to poor solutions corresponding to local minima of the error criterion (as sometimes happens with iterative techniques).
5) A software simulation is easy to write and use.
6) The network can provide a mapping from one set of sample points to another. If the mapping is one to one, an inverse mapping can easily be generated from the same sample points.
7) The clustering versions of GRNN, equations (9)–(12), limit the numbers of nodes and (optionally) provides a mechanism for forgetting old data.

The main disadvantage of GRNN (without clustering) relative to other techniques is that it requires substantial computation to evaluate new points. There are several ways to overcome this disadvantage. One is to use the clustering versions of GRNN. Another is to take advantage of the inherent parallel structure of this network and design semiconductor chips to do the computation. The two in combination provide high throughput and rapid adaptation.

Acknowledgment

The author wishes to express his gratitude to Dr. R. C. Smithson, Manager of the Applied Physics Laboratory, for his support and encouragement, to P. D. Shapiro for demonstrating the use of GRNN for a control problem, and to T. P. Washburne for demonstrating the use of GRNN for the identification problem.

References

[1] L. Ljung, *System Identification—Theory for the User*. Englewood Cliffs, NJ: Prentice-Hall, 1987.
[2] L. Ljung and S. Gunnarsson, "Adaptive tracking in system identification—a survey," *Automatica*, vol. 26, no. 1, pp. 7–22, 1990.
[3] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, Mar. 1990.
[4] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing*, vol. 1, *Foundations*. Cambridge, MA: MIT Press, 1986, ch. 8.
[5] D. F. Specht, "Probabilistic neural networks for classification, mapping, or associative memory," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 1, pp. 525–532, June 1988.
[6] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109–118, 1990.
[7] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281–294, 1989.
[8] D. S. Broomhead and D. Lowe, "Multi-variable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355, 1988.
[9] T. Poggio and F. Girosi, "A theory of networks for approximation and learning," A. I. Memo No. 1140, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, July 1989.
[10] W. T. Miller III, F. H. Glanz, and L. G. Kraft III, "CMAC: An associative neural network alternative to backpropagation," *Proc. IEEE*, vol. 78, pp. 1561–1567, Oct. 1990.
[11] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Statist.*, vol. 33, pp. 1065–1076, 1962.
[12] T. Cacoullos, "Estimation of a multivariate density," *Ann. Inst. Statist. Math*, (Tokyo), vol. 18, no. 2, pp. 179–189, 1966.
[13] D. F. Specht, "Generation of polynomial discriminant functions for pattern recognition," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 308–319, 1967.
[14] D. F. Specht, "A practical technique for estimating general regression surfaces," Lockheed Missiles & Space Company, Inc., Palo Alto, CA, LMSC-6-79-68-6, June 1968; also available as Defense Technical Information Center AD-672505 or NASA N68-29513.
[15] D. F. Specht, "Series estimation of a probability density function," *Technometrics*, vol. 13, no. 2, May 1971.
[16] I. Grabec, *Prediction of a Chaotic Time Series by a Self-Organizing Neural Network*. Dusseldorf, FRG: Dynamics Days, 1990.
[17] L. Breiman, W. Meisel, and E. Purcell, "Variable kernel estimates of multivariate densities," *Technometrics*, vol. 19, no. 2, May 1977.
[18] P. Burrascano, "Learning vector quantization for the probabilistic neural network," *IEEE Trans. Neural Networks*, vol. 2, pp. 458–461, July 1991.
[19] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading, MA: Addison-Wesley, 1974.
[20] M.-L. Tseng, "Integrating neural networks with influence diagrams for multiple sensor diagnostic systems," Ph.D. dissertation, Univ. of California at Berkeley, Aug. 1991.
[21] D. L. Reilly *et al.*, "A neural model for category learning," *Biol. Cybern.*, vol. 45, pp. 35–41, 1982.
[22] V. Cherkassky and H. Lari-Najafi, "Self-organizing neural network for non-parametric regression analysis," in *Proc. Int. Neural Network Conf.*, (Paris, France), vol. 1, pp. 370–374, July 1990.
[23] D. F. Specht and P. D. Shapiro, "Training speed comparison of probabilistic neural networks with back-propagation networks," in *Proc. Int. Neural Network Conf.*, (Paris, France), vol. 1, pp. 440–443, July 1990.
[24] C. G. Atkeson and D. J. Reinkensmeyer, *Neural Networks for Control*, T. Miller *et al.*, Eds. Cambridge, MA: MIT Press, 1990, ch. 11.
[25] J. Farrell *et al.* "Connectionist learning control systems: Submarine depth control," in *Proc. 29th IEEE Conf. Decision and Control*, Dec. 1990.