FUNCTION APPROXIMATION AND TIME SERIES PREDICTION WITH NEURAL NETWORKS

R. D. Jones[a], Y. C. Lee[b], C. W. Barnes,
G. W. Flake, K. Lee, P. S. Lewis,
and S. Qian

University of California
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

Manuscript completed December 7, 1989

ABSTRACT

Neural networks are examined in the context of function approximation and the related field of time series prediction. A natural extension of radial basis nets is introduced. It is found that use of an adaptable gradient and normalized basis functions can significantly reduce the amount of data necessary to train the net while maintaining the speed advantage of a net that is linear in the weights. The local nature of the network permits the use of simple learning algorithms with short memories of earlier training data. In particular, it is shown that a one dimensional Newton's method is quite fast and reasonably accurate.

[a] rdj@lanl.gov

[b] also: Department of Physics and Astronomy and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20740

I. INTRODUCTION

Good internal modeling can be important for process control. Given an input to a plant, control requires some prediction of the output. Furthermore, the prediction must be able to change as the plant ages or changes in some possibly unknown manner. Therefore, any function approximation that models plant behavior must be adaptable. Related to the problem of plant output prediction is the problem of time series prediction. Given the state of the plant at the present, what will the state be at some future time. A selection of plants in which function approximation and time series prediction are important for control is displayed in Fig. 1.

Neural networks have demonstrated an impressive ability to deal with problems of this sort. At present, the most popular net for function approximation is a feedforward backpropagation network (Rumelhart, Hinton, and Williams 1986). This net is illustrated in Fig. 2. The net is composed of input and output layers, and one or more hidden layers of neurons. Information flow is in a single direction, in this case to the right. The output, y_i, of the i^{th} neuron is given by

$$y_i = sig(\Sigma_j W_{ij} y_j + \theta_i) \qquad I-1$$

where y_j is the output of the j^{th} neuron in a layer immediately to the left of the layer in which the i^{th} neuron is located. The sigmoid function, sig, is a rounded Heaviside step function,

$$sig(x) = (1/2)[1 + tanh(x)]. \qquad I-2$$

The form of this function is chosen to mimic, in a rough sense, biological neurons. The weights, W_{ij}, and thresholds, θ_i, are determined by least mean squares minimization. Define a cost function,

$$E = (1/2)\Sigma_{p=1}^{M}[f(\vec{x}_p) - \phi(\vec{x}_p)]^2 \qquad I-3$$

where $\vec{x}_p$ is an input training vector, $f(\vec{x}_p)$ is the training output for the input, $\vec{x}_p$, and $\phi(\vec{x}_p)$ is the network output for the training input, $\vec{x}_p$. The summation is over all training points. We do not rule out the possibility that a training point can be shown to the net more than once. In fact, that is the usual situation. Therefore, M is the number of times that any training point is shown to the net. For convenience, we have assumed a scalar output. There is no fundamental reason that there could not be multiple outputs. The learning algorithm is simply the numerical technique for the minimization of E. Common minimization methods, for instance, are gradient descent, conjugate gradient, and Newton's method (Press, Flannery, Teukolsky, and Vetterling 1986).

Backpropagation networks have had some impressive successes. This class of networks has been able to beat nearly all the traditional methods in the accuracy (Fig.3) of time series prediction (Lapedes and Farber 1987 and 1989). Here the game is, given that a certain number of points of a time series have been sampled in the present and past, what will the value of the time series be at some future time.

Another impressive example is that of control of a backing truck (Fig. 4) (Nguyen and Widrow 1989). Two nets are used in this problem, one to learn how a truck works and the other to learn how to back a truck to a loading dock given that the net knows how a truck works. The truck is only permitted to back, not drive forward.

Backpropagation networks have some problems, however: (1) They require a great deal of training data; (2) Interpolation is poor without a great deal of training; (3) Backpropagation nets are much slower, for comparable accuracy, than the best non-neural methods (Farmer and Sidorowich 1987). In many practical problems large amounts of data are not available. At minimum, a neural net must be able to automatically interpolate and extrapolate from a small data set. Also, in many problems, learning must occur in real time. Slow learning limits the application of neural nets.

FUNCTION APPROXIMATION AND TIME SERIES PREDICTION IS IMPORTANT IN CONTROL

Chemical plants
Cardiac pacemakers
Trains, airplanes, missiles
Sonar, radio, seismic signals
Roulette wheels
SDI systems (FEL, NPB)
Manufacturing (robots, toxic waste minimization, etc.)
Music recording
Battlefield management, global strategies
Epilepsy
Stock market
Power grids
Large physics experiments (SSC, LMF)

FIGURE 1

THE STANDARD FUNCTION APPROXIMATOR IS A BACKPROPAGATION NETWORK.

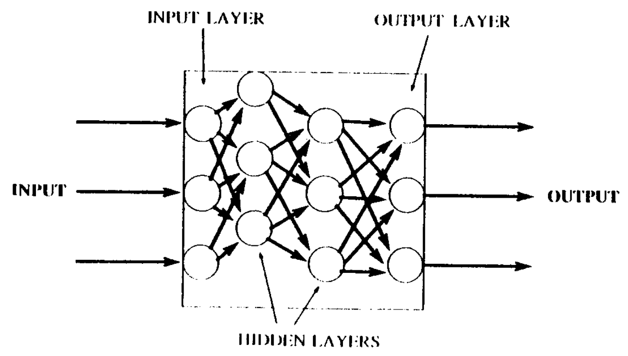


FIGURE 2

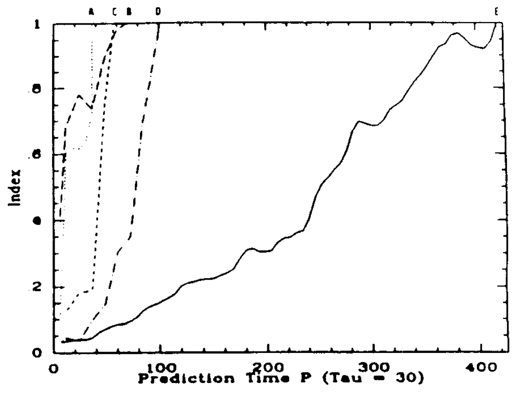
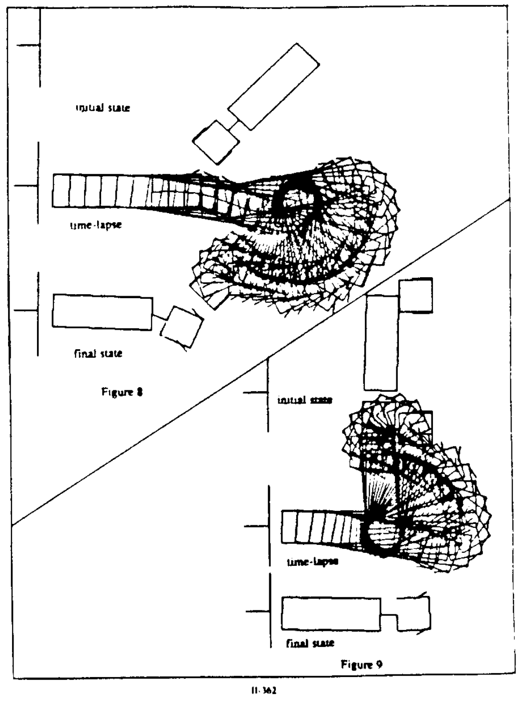


FIGURE 3

Truck backing has been learned by a backpropagation network.



D. Nguyen and B. Widrow. 1989. The truck backer-upper: An example of self-learning in neural networks.

FIGURE 4

Learning speed has been addressed by Moody and Darken (1989). They replaced the backpropagation net of sigmoidal nonlinear elements with a net of the form (see Fig. 5)

$$\phi(\vec{x}) = \Sigma_{j=1}^{N} f_j \rho_j(\vec{x}). \qquad I-4$$

Here, $\rho_j(\vec{x})$, called a radial basis function, is a local function of $(\vec{x} - \vec{x}_j)^2$ and N is the number of basis functions (neurons in the hidden layer). Training is done on the coefficients, f_j. Significant speedup in training is obtained because of this linearity. The net is, however, nonlinear in the inputs. Moody and Darken estimate the reduction in learning rate on the Mackey-Glass equation for comparable accuracy to be a factor of 10^2-10^3. Casdagli (1989) has shown how this net may be used to obtain invariants of a chaotic time series.

Radial basis function nets do not, however, address the problem of excessive training data or poor interpolation. This is illustrated in Fig. 6. The figure is taken from Moody and Darken (1989). To achieve comparable accuracy to the backpropagation benchmark of the Mackey-Glass equation, several more neurons are required. More significantly, the number of training data points required to train the net is greater by a factor of 27.

II. CONNECTIONIST NORMALIZED LINEAR SPLINE NET (CNLS)

A. THE ARCHITECTURE

A natural evolution is to modify radial basis function nets in a manner that improves interpolation and reduces the amount of training necessary for accurate learning (Lee 1989, Jones 1989, and Howell, Barnes, Brown, Flake, Jones, Lee, Qian, Wright 1989). Note the identity,

$$g(\vec{x}) = \frac{\Sigma_{j=1}^{N} g(\vec{x}) \rho_j(\vec{x})}{\Sigma_j \rho_j(\vec{x})}. \qquad II-1$$

Once again, $\rho_j(\vec{x})$ is a localized function of $\vec{x}$ about some $\vec{x}_j$. Hence, $g(\vec{x})$ on the right of Eq. II-1 can be approximated by its Taylor expansion about $\vec{x}_j$. We have then,

$$\phi(\vec{x}) = \Sigma_{j=1}^{N} [f_j + (\vec{x} - \vec{x}_j) \cdot \vec{d}_j] \frac{\rho_j(\vec{x})}{\Sigma_j \rho_j(\vec{x})} \qquad II-2$$

for an approximation to $g(\vec{x})$. This net differs from the radial basis function net in two ways, the use of normalization and also a linear term, $(\vec{x} - \vec{x}_j) \cdot \vec{d}_j$. The use of a normalization term was suggested but not pursued by Moody and Darken (1989). The addition of these two terms is responsible for the reduction in the amount of training data needed to obtain reasonable approximations. As in the case with radial basis functions, the training of f_j and $\vec{d}_j$ is linear and hence very fast. The widths of the basis functions can also be trained. This training is nonlinear. We will show, however, for the learning algorithm we adopt that this training can also be very fast. In some

SIGNIFICANT INCREASE IN LEARNING SPEED CAN BE ACHIEVED BY USING LOCAL BASIS FUNCTIONS.

RADIAL BASIS FUNCTIONS

Speedups of 1000 have been estimated.

$$\boxed{f(x) \sim \phi(x) = \Sigma_j\ f_j\ \rho_j(x)}$$

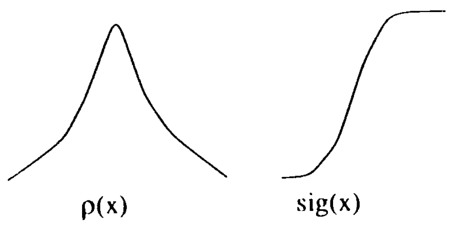
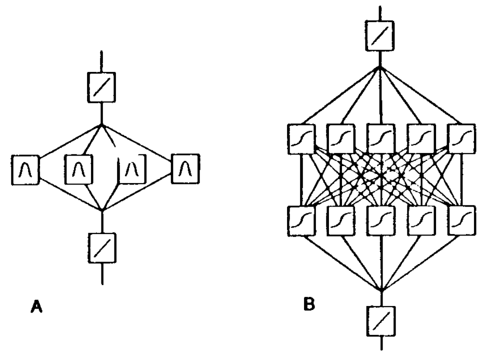


FIGURE 5



For comparable accuracy, however, radial basis functions require many more neurons and much more training data.

FIGURE 6

There are 10 training points for every neuron. 27 times the the number of training points are needed for comparable accuracy with the radial basis function net than for the standard backpropagation net.

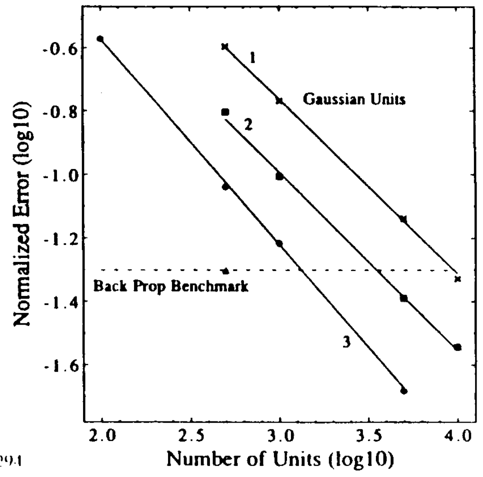


Figure 5: Comparison of prediction accuracy vs. number of internal units for four methods: (1) first nearest neighbor, (2) adaptive units (one unit per training vector), (3) self-organizing units (ten training vectors per processing unit), and (A) backpropagation. The methods are described in the text. For backpropagation, the abscissa indicates the number of training vectors. The horizontal line associated with (A) is provided for visual reference and is not intended to suggest a scaling law. In fact, the scaling law is not known.

J. Moody and C. J. Darken. 1989. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1, 281-294

Can the speed advantages of the radial basis function technique be retained while reducing the amount of data required to train the net?

applications the components of the input training vector are of different types. For example, one might want to include both slope and curvature information in the input vector $\vec{x}$. A radial function of $(\vec{x}-\vec{x}_p)^2$, which weights every component equally might not be appropriate. In these situations differential weighting of the components can be employed. The basis function will then be wider in some directions than others. Care must be used when training the widths differentially as multi-humped basis functions can result.

B. THE LEARNING ALGORITHM

We will not attempt a rigorous justification of our learning algorithm. Rather, we will try to motivate the algorithm heuristically and by analogy.

Learning algorithms can be either online or offline. Offline algorithms attempt to calculate weights without any reference to time ordering of the training data. Thus, all the training data must be collected before training can start. Online algorithms, on the other hand, attempt to modify weights as information in the form of training data flows in. Online algorithms are able to handle varying amounts of training data and are able to modify the system in the presence of drift in the conditions. This is very difficult with an offline algorithm. Additionally, online algorithms are able to handle amounts of input data that would severely tax memory storage capacities if an offline method were used. Most neural nets are trained, therefore, with online methods.

Online methods themselves come in two extremes. The method can remember all the data that it has been shown up to the present or it can only be aware of the training set it is being shown at the present. Conjugate gradient learning and multidimensional Newton's method fall into the former category. We will use a method that falls into the latter category. Since we will use less information, we will pay a price in accuracy, but this is compensated by speed and simplicity.

Before we minimize Eq. I-3 to find f_j and $\vec{d}_j$, consider the alternate problem, the inversion of

$$y_i(\vec{x}_p) = \Sigma_{j=1}^{N} W_{ij} u_j(\vec{x}_p) \qquad II-3$$

to find the matrix, W_{ij}, given the set of training vectors, y and $\vec{x}$. Here, u is similar to the ρ vectors in Eq. II-2 in that it is a localized vector function of the training input, $\vec{x}_p$. We require the u vectors to be normalized.

$$1 = \Sigma_{j=1}^{N} u_j$$

W_{ij} might correspond to either the f_j or $\vec{d}_j$ quantities in Eq. II-2. The vector, $y(x_p)$, corresponds to the target value, $f(x_p)$. We would like Eq. II-3 to be true for any training pair, $[\vec{x}_p, y(\vec{x}_p)]$. This is, in general, not possible when the number of training points exceeds N, the number of dimensions of u. We will show, however, a very good approximation is possible. Eq. II-3 can be rewritten

$$\Sigma_j W_{ij} u_j(\vec{x}_p) = \Sigma_j W_{ij}^0 u_j(\vec{x}_p) + [y_i(\vec{x}_p) - \Sigma_j W_{ij}^0 u_j(\vec{x}_p)] \frac{\Sigma_k u_k^2(\vec{x}_p)}{\Sigma_k u_k^2(\vec{x}_p)} \qquad II-4$$

where W_{ij}^0 is some arbitrary guess for W_{ij}. If we do not consider information from previous training points, then the best information available is that the change in W lies in the direction of the current vector $u(\vec{x}_p)$ where $\vec{x}_p$ is the current training point. We have, then

$$W_{ij} = W_{ij}^0 + [y_i(\vec{x}_p) - \Sigma_l W_{il}^0 u_l(\vec{x}_p)] \frac{u_j(\vec{x}_p)}{\Sigma_k u_k^2(\vec{x}_p)}. \qquad II-5$$

(Note added in proof: It has come to our attention that if one replaces $u(\vec{x}_p)$ with $\vec{x}_p$ in II-5 then the $\alpha - LMS$ algorithm of Widrow is recovered.) All the training algorithms we use are based on Eq. II-5. Equation II-5 is equivalent to a one dimensional Newton's method for finding roots. To see this, define

$$g(W) = \Sigma_j W_{ij} u_j - y. \qquad II-6$$

Then Eq. II-5 can be written

$$W_{ij} = W_{ij}^0 - g(W^0) \frac{\nabla g(W^0)}{\nabla g(W^0) \cdot \nabla g(W^0)} \qquad II-7$$

where the gradient operation is with respect to W^0. This is Newton's method for finding the root of g along a line parallel to the gradient of g (Press, Flannery, Teukolsky, and Vetterling 1986). Equation II-5 is also equivalent to gradient descent learning (Press, Flannery,Teukolsky, and Vetterling 1986) with a variable learning rate, $1/\Sigma_j \rho_j^2$.

The solution obtained with Eq. II-5 can be compared with the solution obtained from a least mean squares minimization. In analogy with Eq. I-3, define the cost function

$$I_i = (1/2) < [y_i - \Sigma_{j=1}^N W_{ij} u_j]^2 >, \qquad II-8$$

where the averaging is defined

$$< h >= (1/M) \Sigma_{p=1}^M h(\vec{x}_p). \qquad II-9$$

Here, M is the number of times a training point is shown to the net. M is greater than or equal to the number of training points. Minimizing Eq. II-8 with respect to W_{ij} yields

$$< y_i u_j >= \Sigma_k W_{ik}^* < u_k u_j > . \qquad II-10$$

Here, W^* is the value of W which minimizes Eq. II-8. In analogy with Eq. II-5, Eq. II-10 can be written

$$W_{ij}^* = W_{ik}^0 + \Sigma_k [< y_i u_k > - W_{ij}^0 < u_j u_k >] < u_k u_j >^{-1} \qquad II-11$$

where once again W_{ij}^0 is some arbitrary guess for W_{ij}. Since Eq. II-5 is shown all the training points, the solution of Eq. II-5 can be obtained by averaging.

$$W_{ij} = W^0_{ij} + \left[< \frac{y_i u_j}{\Sigma_k u_k^2} > - \Sigma_l W^0_{il} < \frac{u_l u_j}{\Sigma_k u_k^2} > \right]. \qquad II-12$$

Comparison of Eqs. II-11 and II-12 yields the requirements for the convergence of Eq. II-5 to the least mean squares solution,

$$\Sigma_k < \frac{y_i u_k}{\Sigma_k u_k^2} >< u_k u_j >=< y_i u_j > \qquad II-13$$

and

$$< \frac{u_i u_j}{\Sigma_k u_k^2} >= \delta_{ij}. \qquad II-14$$

A sufficient condition for these requirements to be satisfied is that the components of u should be very localized in $\vec{x}_p$. The normalization of u is very important for this condition to work.

In practice, Eq. II-5 is iterated as data is presented to the net. This can be represented as

$$W^{p+1}_{ij} = W^p_{ij} + [y_i(\vec{x}_p) - \Sigma_l W^p_{il} u_l(\vec{x}_p)] \frac{u_j(\vec{x}_p)}{\Sigma_k u_k^2(\vec{x}_p)}. \qquad II-15$$

If a very good approximation to Eq. II-3 exists for all $\vec{x}_p$ and if the vectors, $u(\vec{x}_p)$ span the space, then it can be shown that Eq. II-5 converges to the solution of Eq. II-13. To see this, subtract Eq. II-15 from W_{ij} and use Eq. II-3 to obtain

$$W^{p+1}_{ij} - W_{ij} = \Sigma_l \left[\delta_{jl} - \frac{u_j u_l}{\Sigma_k u_k^2} \right] [W^p_{il} - W_{il}]. \qquad II-16$$

We see that when a good solution of Eq. II-3 exists, that Eq. II-15 reduces to a projection operator. Therefore, if the u vectors span the space, then W^p_{ij} converges to W_{ij}.

The network of interest, Eq. II-2, differs from the model net, Eq. II-3, in two important ways: (1) there are two sets of linear weights to train instead of one, and (2) there are two sets of qualitatively different basis vectors instead of one. We can reduce the approximation, Eq. II-2, to two problems of the form of Eq. II-3. If we assume that we know the optimum gradient, $\vec{d}$, in Eq. II-2 then the quantities f_j play the role of W_{ij} and the quantities $\rho_j / \Sigma_j \rho_j$ play the role of u_j. In analogy with Eq. II-15 we have then

$$f^{p+1}_j = f^p_j + [g(\vec{x}_p) - \Sigma_l (f^p_l + (\vec{x}_p - \vec{x}_l) \cdot \vec{d}_l)] \frac{\rho_j(\vec{x}_p) \Sigma_k \rho_k(\vec{x}_p)}{\Sigma_k \rho_k^2(\vec{x}_p)} \qquad II-17$$

where $\vec{d}_l$ is the optimum value. If we assume that we know f_l but not $\vec{d}_l$ then once again the problem is of the form of Eq. II-3 and we have in analogy to Eq. II-15

$$\vec{d}^{\,p+1}_j = \vec{d}^{\,p}_j + [g(\vec{x}_p) - \Sigma_l (f_l + (\vec{x}_p - \vec{x}_l) \cdot \vec{d}^{\,p}_l)] \frac{(\vec{x}_p - \vec{x}_j) \rho_j(\vec{x}_p) \Sigma_k \rho_k(\vec{x}_p)}{\Sigma_k (\vec{x} - \vec{x}_k)^2 \rho_k^2(\vec{x}_p)}. \qquad II-18$$

Equations II-17 and II-18 can be iterated for f_j^{p+1} and $\vec{d}_j^{\,p+1}$ if we have some approximation for $\vec{d}_l$ in Eq. II-17 and f_l in II-18. If we approximate these quantities by $\vec{d}_l^{\,p}$ and f_l^p, respectively, then we have an explicit iteration scheme

$$f_j^{p+1} = f_j^p + [g(\vec{x}_p) - \phi(\vec{x}_p)]\frac{\rho_j(\vec{x}_p)\Sigma_k\rho_k(\vec{x}_p)}{\Sigma_k\rho_k^2(\vec{x}_p)} \qquad II-19$$

and

$$\vec{d}_j^{\,p+1} = \vec{d}_j^{\,p} + [g(\vec{x}_p) - \phi(\vec{x}_p)]\frac{(\vec{x}_p - \vec{x}_j)\rho_j(\vec{x}_p)\Sigma_k\rho_k(\vec{x}_p)}{\Sigma_k(\vec{x} - \vec{x}_k)^2\rho_k^2(\vec{x}_p)}. \qquad II-20$$

Unfortunately, explicit iteration schemes tend to be unstable. Tests indicate that this scheme, in particular, is unstable. This problem can be corrected by implicitly approximating $\vec{d}_l$ and f_l with $\vec{d}_l^{\,p+1}$ and f_l^{p+1}. Doing this and solving for the advanced quantities yields

$$f_j^{p+1} = f_j^p + \frac{1}{3}[g(\vec{x}_p) - \phi(\vec{x}_p)]\frac{\rho_j(\vec{x}_p)\Sigma_k\rho_k(\vec{x}_p)}{\Sigma_k\rho_k^2(\vec{x}_p)} \qquad II-21$$

and

$$\vec{d}_j^{\,p+1} = \vec{d}_j^{\,p} + \frac{1}{3}[g(\vec{x}_p) - \phi(\vec{x}_p)]\frac{(\vec{x}_p - \vec{x}_j)\rho_j(\vec{x}_p)\Sigma_k\rho_k(\vec{x}_p)}{\Sigma_k(\vec{x} - \vec{x}_k)^2\rho_k^2(\vec{x}_p)}. \qquad II-22$$

The only difference between the explicit and implicit schemes is that the error term in the implicit scheme is multiplied by a factor of 1/3. In all the cases we have examined, this is sufficient to insure stability. The correction terms in Eqs. II-17 and II-18 may be multiplied by a constant factor. This corresponds to changing the learning rate. Varying this factor from 1 to infinity changes the factor of 1/3 in Eqs. II-21 and II-22 to 1/2 thus slightly increasing the learning rate in the implicit scheme.

The architecture, Eq. II-2, and the learning algorithm, Eqs. II-21 and II-22, form the backbone of the network. All the examples of the next section are based on these three equations or on slight variations of them.

C. EXAMPLES

Following Lapedes and Farber (1987) and Moody and Darken (1989) we test the network on the evolution of the logistic map. Here, the network must approximate the function

$$x_{n+1} = 4x_n(1 - x_n).$$

The input to the net is x_n and the output is x_{n+1}. The results are displayed in Fig. 7. As in the case of Lapedes and Farber (1987) and Moody and Darken (1989), we use five neurons (basis functions) in our hidden layer. These authors used, in addition, a linear connection between the input neuron and the output neuron. There are a total of 10 weights total to be trained in our net. This is to be compared with 17 weights in the backpropagation net of Lapedes and Farber and 15 for Moody and Darken's net. The previous authors used 1000 input-output pairs to train their nets. Initially, we use 4. The locations of the four training points are marked with diamonds and arrows in Fig. 7. The centers of the basis functions are marked with long dashes on the abscissa of the top plot in Fig. 7. Conjugate gradient training was used in the backpropagation and radial basis function nets, while we have used the one dimensional Newton's method of Eqs. II-21 and II-22. As an initial guess for the weights we chose $f_j = g(x_j)$ and $d_j = 0$. The functional form for the basis functions were taken to be Gaussian, $exp(-\beta(x - x_j)^2)$, where $\beta = 1/0.5^2$. The approximation based on this initial guess is displayed. The four training points were shown to the net randomly and in this training run the net saw a training point 4000 times (1000 times, on average, per training pair).

The net performs well even when presented with only four training points. It can be seen that a smooth and reasonable interpolation obtains. The worst fit occurs in the lower left portion of the parabola where the net is forced to extrapolate. The error here is 5 to 10 percent. Prediction of a novel time series is displayed in the middle plot. It can be seen from the absolute error curve on the training points that most of the learning occurred in the first 20 training cycles. Slow improvement is seen after that.

If the net is trained on 100 points, the prediction accuracy improves. This can be seen in Fig. 8. Once again the learning occurs on a fast and a slow time scale.

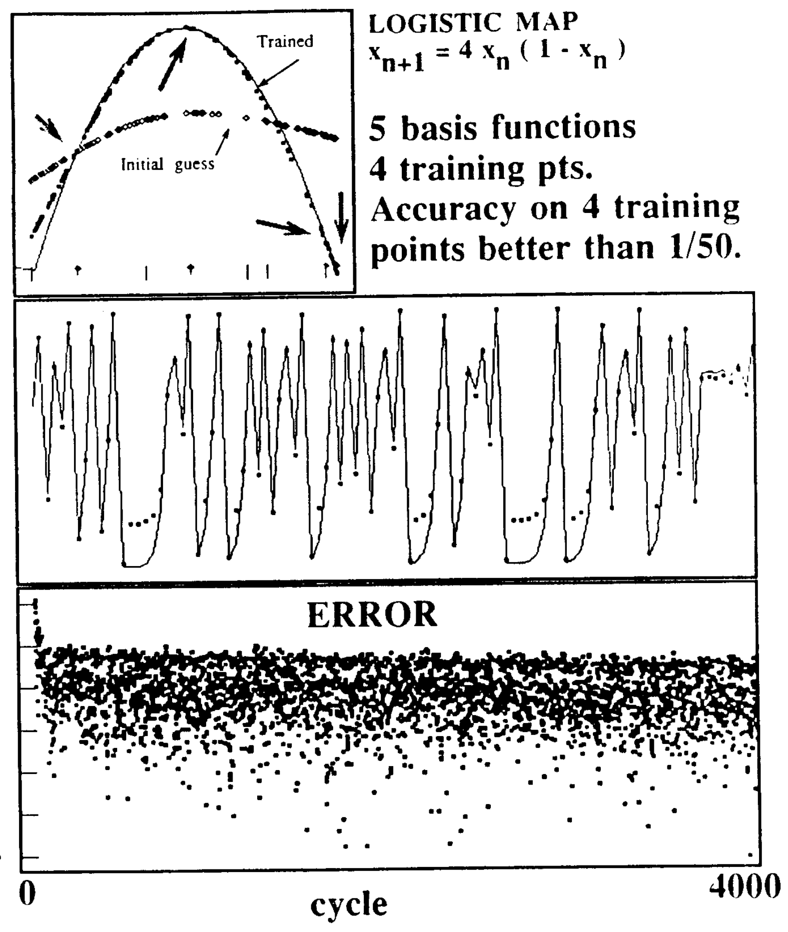
Training on the widths, β_j, improves accuracy even more. This is displayed in Fig. 9. The total prediction error drops to less than a percent and becomes comparable to the error of approximately 0.5 percent using backpropagation or radial basis functions with conjugate gradient learning (Moody and Darken 1989). With training on widths, the total number of adjustable weights in the net increases to 15 which is comparable to the backpropagation and radial basis function tests. The total number of training points, here, is 100 compared with 1000 used by the other nets. Training took a few minutes of SUN 3/75 time as compared with an hour of SUN 3/50 time for the other two nets.

The effects of the normalization and gradient terms can be seen in Figs. 10 and 11. Without these terms, the prediction is about a factor of five worse than the prediction with the terms (Fig. 7). Here, four training points were used.

Performance on a non-smooth map is displayed in Fig. 12. Here, 10 basis functions and 100 training points were used to predict a tent map. The widths of the basis functions were chosen to be $\beta = 5/0.5^2$ and only f_j and d_j were trained. Not surprisingly, the net rounds off the point of the tent.

Finally, we look (Fig. 13) at a system slightly more complicated than a one dimensional map, the Mackey-Glass equation (Mackey and Glass, 1977). The problem, here, is to find the smallest number of basis functions and training points which give reasonable predictions. In this case, we choose four input dimensions ($\tau = 17$). Sampling period was six and the prediction was on the point at $t + 6$. Reasonable predictions (worst

The learning algorithm performs well even when presented with minimal information.

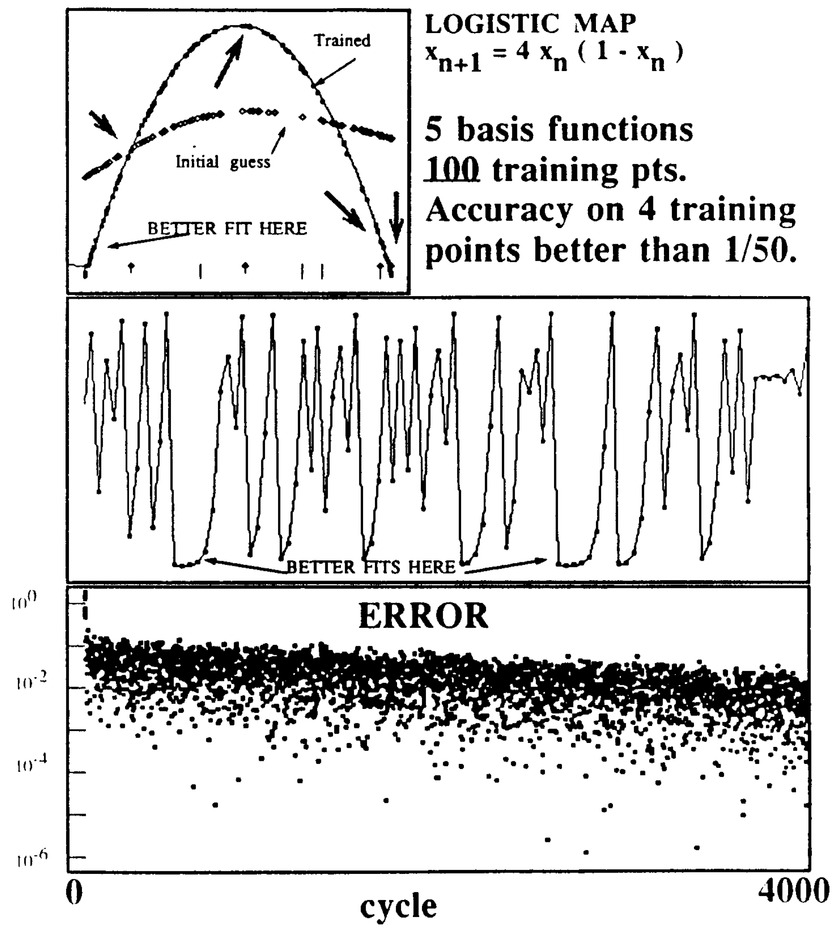


LOGISTIC MAP
$x_{n+1} = 4\, x_n\, (1 - x_n)$

5 basis functions
4 training pts.
Accuracy on 4 training points better than 1/50.

FIGURE 7

Increasing the information presented to the net improves the accuracy.



LOGISTIC MAP
$x_{n+1} = 4\, x_n\, (1 - x_n)$

5 basis functions
<u>100</u> training pts.
Accuracy on 4 training points better than 1/50.

FIGURE 8

ACCURACY CAN BE SIGNIFICANTLY IMPROVED BY TRAINING ON WIDTHS

LOGISTIC MAP
$x_{n+1} = 4\,x_n\,(1 - x_n)$

5 basis functions
<u>100</u> training pts.

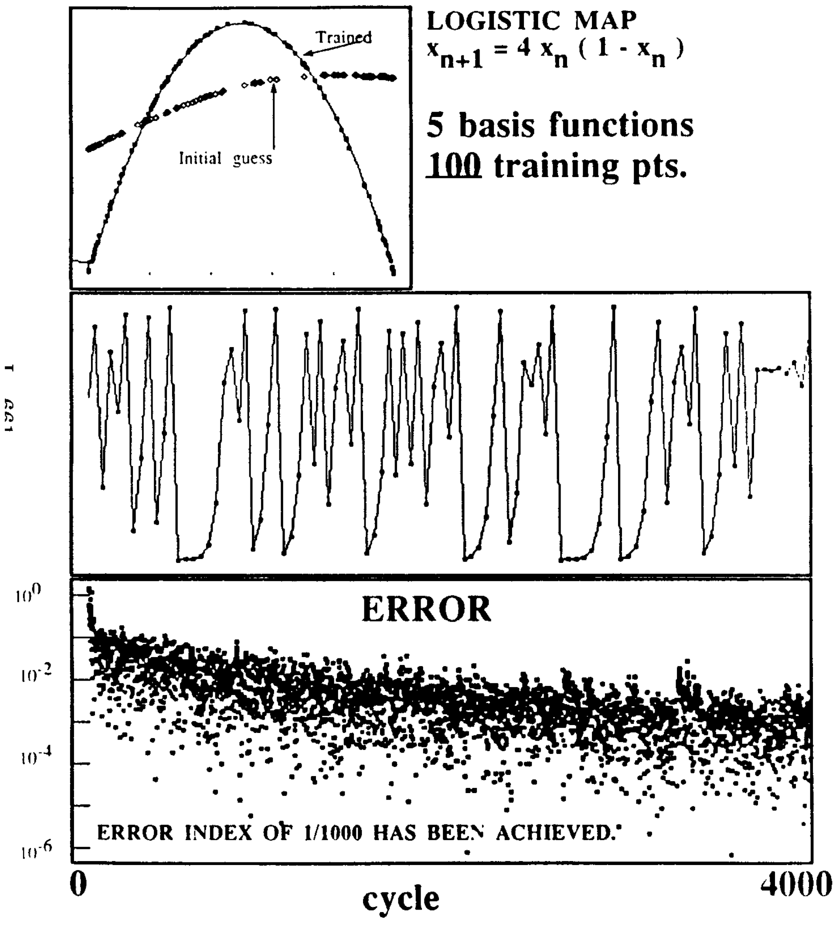


FIGURE 9

Without basis function normalization a poorer fit obtains.

LOGISTIC MAP
$x_{n+1} = 4\,x_n\,(1 - x_n)$

5 basis functions
4 training pts.
Accuracy on 4 training points better than 1/10.

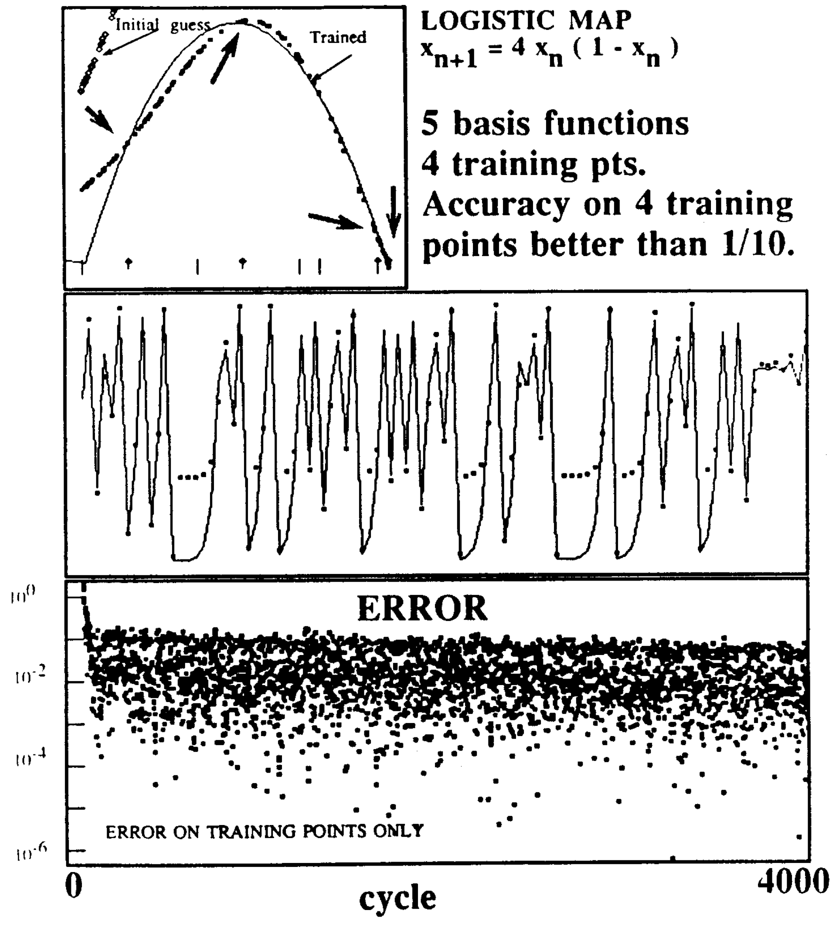


FIGURE 10

Without basis function normalization and gradient training an even poorer fit obtains.

LOGISTIC MAP
$x_{n+1} = 4\, x_n\, (1 - x_n)$

5 basis functions
4 training pts.
Accuracy on 4 training points worse than 1/10.

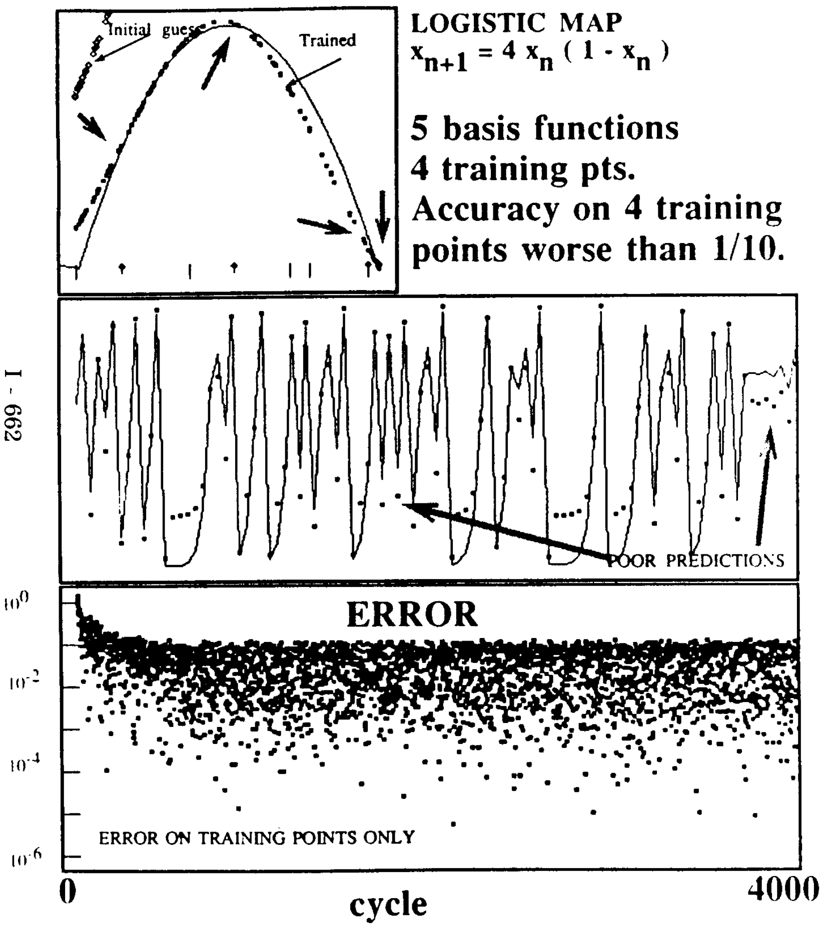


FIGURE 11

Reasonable fit obtains to non-smooth maps

TENT MAP

10 basis functions
100 training pts.
Accuracy on 100 training points better than 1/10.

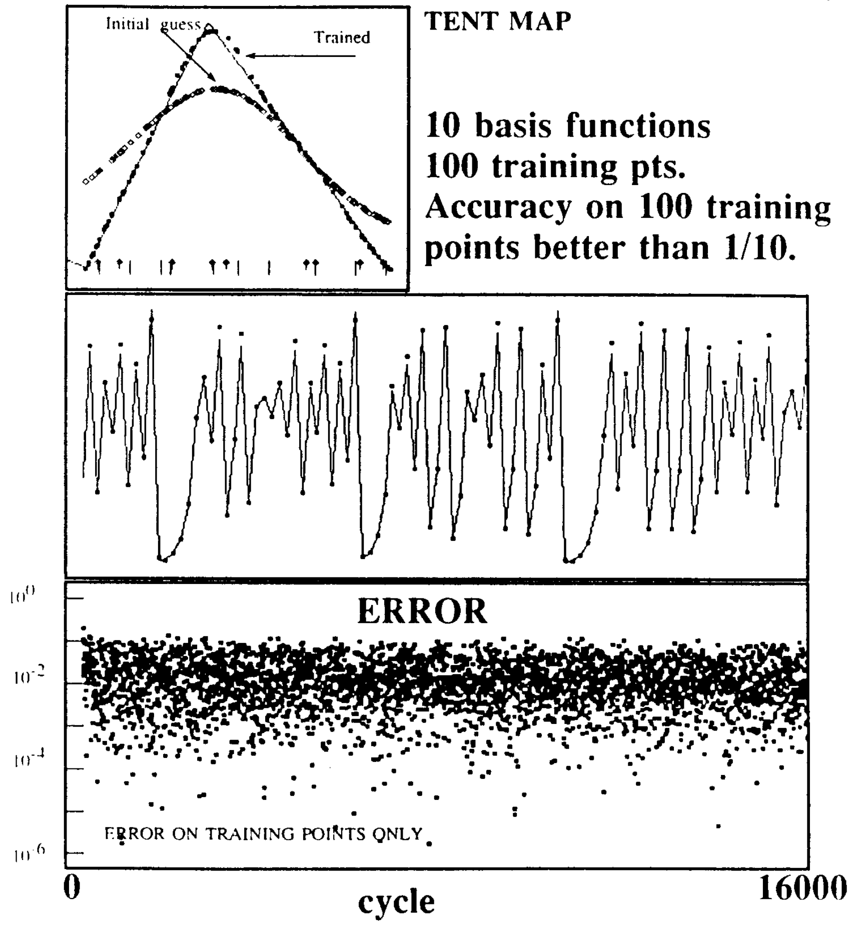
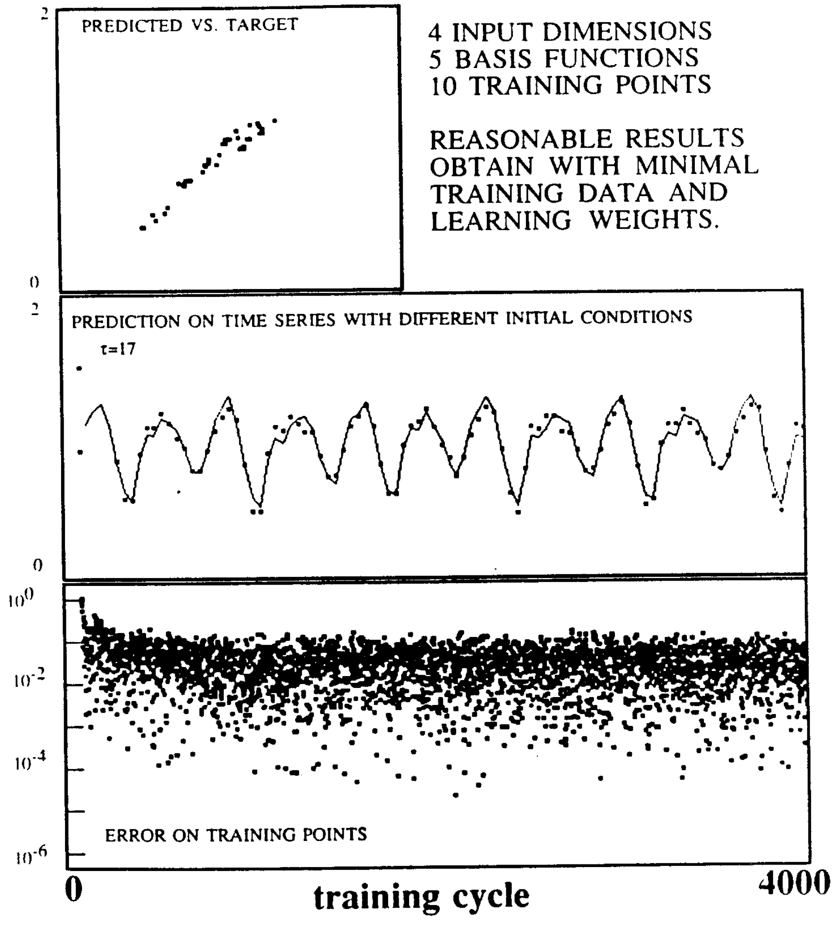


FIGURE 12

MACKEY-GLASS EQUATION



4 INPUT DIMENSIONS
5 BASIS FUNCTIONS
10 TRAINING POINTS

REASONABLE RESULTS OBTAIN WITH MINIMAL TRAINING DATA AND LEARNING WEIGHTS.

FIGURE 13

error of approximately 10 percent) are obtained with five basis functions and only ten training points.

III. SUMMARY

We have developed a neural net architecture which has the speed advantage of radial basis functions, but which has greater ability to generalize. The amount of data necessary to train the net is reduced significantly. The network architecture permits the use of a simple learning algorithm (essentially a one dimensional Newton's method). The architecture and learning algorithm are concisely presented in Eqs. II-2, II-21, and II-22. The network has been applied to a number of simple examples, some of which have been presented in Sec. II. and is being applied to a number of real world problems, such as free electron laser design surface approximation, control of a negative ion source (Howell et al. 1989), and to transient detection in sonar signals. Work is underway to modify the net to handle noisy data.

REFERENCES

Casdagli M. 1989. Nonlinear prediction of chaotic time series. *Physica D*, **35**, 335-356.

Farmer J. D. and J. J. Sidorowich. 1987. Predicting chaotic time series. *Physical Review Letters,* **59**, 845.

Howell, J. A., C. W. Barnes, S. K. Brown, G. W. Flake, R. D. Jones, Y. C. Lee, S. Qian, and R. M. Wright 1989. Control of a negative-ion accelerator source using neural networks. To be published in ***Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems*** Vancouver, B.C., Canada October 30-November 3, 1989. Los Alamos report LA-UR-89-3597.

Jones, R. D. 1989. Function approximation and time series prediction with neural networks. Talk presented at the Center for Nonlinear Studies, Los Alamos National Laboratory, Los Alamos, New Mexico on September 27, 1989.

Lapedes, A. S. and R. Farber. 1987. ***Nonlinear signal processing using neural networks: Prediction and system modeling.*** Technical Report, Los Alamos National Laboratory, Los Alamos, New Mexico 87545.

———. 1988. How neural nets work. ***In: Neural Information Processing Systems,*** ed. D. Z. Anderson, 442-456. American Institute of Physics.

Lee, Y. C. 1989. Neural networks with memory for intelligent computations. *In:* Proceedings of the 13th Conference on the Numerical Simulation of Plasmas. Santa Fe, New Mexico, September 17-20, 1989.

Mackey, M. C. and L. Glass, Oscillation and chaos in physiological control systems, Science 197 (1977) 287.

Moody J. and C. J. Darken. 1989. Fast learning in networks of locally tuned processing units. ***Neural Computation,*** **1**, 281-294.

Nguyen, D. and B. Widrow. 1989. The truck backer-upper: An example of self-learning in neural networks. Proceedings IJCNN, Washington (June 1989), pp II-357-363.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. 1986. ***Numerical Recipes.*** Cambridge: Cambridge University Press.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. ***In:*** Parallel Distributed Processing, **1**, eds. D. E. Rumelhart and J. L. McClelland, 318-362.