A vector matrix real time backpropagation algorithm for recurrent neural networks that approximate multi-valued periodic functions

Peter Stubberud
University of Nevada, Las Vegas, peter.stubberud@unlv.edu

Follow this and additional works at: https://digitalscholarship.unlv.edu/ece_fac_articles

Citation Information
https://digitalscholarship.unlv.edu/ece_fac_articles/141
A VECTOR MATRIX REAL TIME RECURSIVE BACKPROPAGATION ALGORITHM FOR RECURRENT NEURAL NETWORKS THAT APPROXIMATE MULTI-VALUED PERIODIC FUNCTIONS

PETER STUBBERUD
Department of Electrical and Computer Engineering
University of Nevada Las Vegas
4505 Maryland Parkway
Las Vegas, NV 89154-4026, USA
stubber@ee.unlv.edu

Received

Unlike feedforward neural networks (FFNN) which can act as universal function approximators, recursive, or recurrent, neural networks can act as universal approximators for multi-valued functions. In this paper, a real time recursive backpropagation (RTRBP) algorithm in a vector matrix form is developed for a two layer globally recursive neural network that has multiple delays in its feedback path. This algorithm has been evaluated on two GRNNs that approximate both an analytic and nonanalytic periodic multi-valued function that a feedforward neural network is not capable of approximating.

1. Introduction

The terms, recursive neural network (RNN) and recurrent neural network (RNN), can refer to any neural network that uses feedback within its architecture. A locally recursive, or recurrent, neural network refers to a neural network that has recursive, or recurrent, neurons which are interconnected nonrecursively. A globally recursive, or recurrent, neural network (GRNN) refers to a neural network that has feedforward, or nonrecursive, neurons which are interconnected recursively. The globally recursive neural network discussed in this paper is the two layer globally recursive neural network shown in Figure 1 where \( x_0(n), ..., x_L(n) \) are the network’s nonrecursive inputs, \( y_1(n), ..., y_P(n) \) are the network’s outputs, \( y_1(n-1), ..., y_P(n-M) \) are the network’s recursive inputs feedback from the output and the functions, \( f_1(\cdot) \) and \( f_2(\cdot) \), are nonrecursive activation functions.

A feedforward neural network (FFNN) with two layers of neurons can uniformly approximate any continuous function on any compact set arbitrarily well regardless of the activation function in the first layer of neurons [1]. Because the GRNN in Figure 1 is identical to such a FFNN except that previous outputs are included as inputs, this GRNN architecture can also act as universal function approximators. Also, because this GRNN includes previous outputs as inputs, this architecture can also approximate multi-valued functions that FFNNs cannot approximate. Because multi-valued functions are required to model systems with hysteresis, RNNs and not FFNNs have the potential to act as uni-
universal system approximators as well as universal function approximators. Also, a smaller GRNN architecture can also potentially provide the same functionality as a larger FFNN in the same way that infinite impulse response (IIR) filters can replace longer finite impulse response (FIR) filters [2].

Neural networks are often trained using gradient descent algorithms which are algorithms that iteratively uses the gradient of the cost function or an estimate of the gradient of the cost function to determine a cost function’s minimum. Gradient descent algorithms for training recursive neural networks can generally be classified as either backpropagation through time (BPTT) algorithms [3;4;5;6;7] or as real time recurrent learning (RTRL), or real time recursive backpropagation (RTBP), algorithms [6;7;8;9;10;11]. BPTT algorithms compute the gradient for a RNN by transforming the RNN into an equivalent FFNN and then training the equivalent FFNN. These algorithms require storage of multiple network states and some of these algorithms require epoch training and teacher forcing [12;13;14;15]. RTBP algorithms compute the gradient for a RNN at each sample point as the RNN continues to operate, and many variations of the algorithm exist [7]. Several RTRB algorithms use teacher forcing to reduce computational requirements [11;12;16], and several other RTRB algorithms use other methods to reduce computational requirements [17;18;19;20;21]. Several training algorithms combine techniques from BPTT and RTBP [22;23].

In this paper, a real time recursive backpropagation (RTRBP) algorithm is developed for the two layer globally recursive neural network shown in Figure 1. Section 2 of this paper develops a mathematical description of the neural network shown in Figure 1. Section 3 contains the derivation of the RTRBP algorithm in a vector matrix form. This training algorithm was briefly summarized in [24] and was used to restore text images for improving optical character recognition (OCR) accuracy. The algorithm is fully developed in Section 3 of this paper. In Section 4, the RTRBP algorithm is applied to a GRNN

Figure 1. Two layer globally recursive neural network.
which approximates an analytic multi-valued function that a feedforward neural network
is not capable of approximating and a nonanalytic periodic multi-valued function that
a feedforward neural network is not capable of approximating.

2. Two Layer Globally Recursive Neural Network

The two layer GRNN discussed in this paper is equivalent to a two layer FFNN
where some of the outputs have been delayed and then feedback and used as inputs to the
neural network. Figure 1 shows the block diagram of the two layer GRNN discussed in
this paper. In Figure 1, the network’s nonrecursive inputs, \( x_0(n), \ldots, x_L(n) \), can repre-
sent \( L+1 \) simultaneous inputs, or by defining \( x_k(n) \) as \( x(n-k) \), the nonrecursive inputs can
represent sequential samples from a single input. Other combinations of multiple inputs
and sequential samples from one or more of these inputs are possible. The network’s
(MF) recursive inputs, \( y_1(n-1), y_2(n-1), \ldots, y_F(n-M) \), are the first \( F \) of the network’s
\( P \) outputs, \( y_1(n), \ldots, y_P(n) \), that have been delayed up to \( M \) samples and feedback to the
input. In general, \( F \leq P \), and all of the outputs used as recursive inputs can be ordered as
the network’s first \( F \) outputs without loss of generality. The adjustable weights, \( a_{10}, \ldots, a_{NL} \),
are the weights that scale the nonrecursive inputs, and the adjustable weights, \( b_{11}, \ldots, b_{NM}(MF) \),
are the weights that scale the recursive inputs. In particular, the weight, \( a_{rc} \), multiplies the
nonrecursive input, \( x_c(n) \), and the weight, \( b_{rc} \), multiplies the recursive input,
\( y_{[(c-1)mod(F)+1][n-ceil(c/F)]} \), where \( ceil(z) \) rounds \( z \) to the nearest integer to-
wards infinity. These products are summed such that the output, \( o_r(n) \), of the \( r \)th neuron
in the first layer of neurons is

\[
o_r(n) = f_1 \left[ s_r(n) \right]
\]

where

\[
s_r(n) = \sum_{k=0}^{L} a_{rk}x_k(n) + \sum_{i=1}^{M} \sum_{k=1}^{F} b_{rl[(i-1)F+k]}y_k(n-i).
\]  (1)

and the function, \( f_1(\ast) \), is a nonrecursive activation function. An activation function can
be defined as a continuous nondecreasing function that maps the input, \((-\infty, \infty)\), to \([\alpha, \beta]\)
where \( \alpha \) and \( \beta \) are finite nonnegative constants. The adjustable weights, \( v_{1l}(n), \ldots, v_{PL}(n) \),
are the weights that scale the outputs from the first layer of neurons. In particu-
lar, the weight, \( v_{rc} \), multiplies the output, \( o_c(n) \), from the \( c \)th neuron in the first layer of
neurons. The resulting product is summed in the \( r \)th neuron in the second layer of neu-
rons such that this sum, \( q_r(n) \), is

\[
q_r(n) = \sum_{l=1}^{N} v_{rl}(n)f_1 \left[ s_l(n) \right].
\]

The \( r \)th output, \( y_r(n) \), of the GRNN and the output of the \( r \)th neuron in second layer of
neurons is
where the function, \( f_2(\cdot) \), is a nonrecursive activation function. The error signals, \( e_1(n) \), ..., \( e_P(n) \), that are fed to the training algorithm are the difference between the training signals, \( d_1(n) \), ..., \( d_P(n) \), and the GRNN’s output signals; that is, the \( r \)th error signal, \( e_r(n) \), is \( e_r(n) = d_r(n) - y_r(n) \).

The states of the GRNN in Figure 1 can also be written in matrix form by defining the nonrecursive input vector, \( \mathbf{X}(n) \), as

\[
\mathbf{X}(n) = \begin{bmatrix} x_0(n) & x_1(n) & \cdots & x_L(n) \end{bmatrix}^T,
\]

the output vector, \( \mathbf{Y}(n) \), as

\[
\mathbf{Y}(n) = \begin{bmatrix} y_1(n) & y_2(n) & \cdots & y_P(n) \end{bmatrix}^T,
\]

the output vector, \( \mathbf{Y}_F(n) \), that is feedback, as

\[
\mathbf{Y}_F(n) = \begin{bmatrix} y_1(n) & y_2(n) & \cdots & y_F(n) \end{bmatrix}^T
\]

where \( F \leq P \) and the recursive input vector, \( \mathbf{Y}_f(n) \), as

\[
\mathbf{Y}_f(n) = \begin{bmatrix} y_1(n) & \cdots & y_{MF}(n) \end{bmatrix}^T = \begin{bmatrix} \mathbf{Y}_F^T(n-1) & \mathbf{Y}_F^T(n-2) & \cdots & \mathbf{Y}_F^T(n-M) \end{bmatrix}^T.
\]

By combining the nonrecursive input vector, \( \mathbf{X}(n) \), and the recursive input vector, \( \mathbf{Y}_f(n) \), the network’s input vector, \( \mathbf{U}(n) \), can be defined as

\[
\mathbf{U}(n) = \begin{bmatrix} \mathbf{X}(n) \\ \mathbf{Y}_f(n) \end{bmatrix}.
\]

By defining a weight matrix, \( \mathbf{A}(n) \), for the nonrecursive input vector, \( \mathbf{X}(n) \), as

\[
\mathbf{A}(n) = \begin{bmatrix} a_{10}(n) & \cdots & a_{1L}(n) \\ \vdots & \ddots & \vdots \\ a_{N0}(n) & \cdots & a_{NL}(n) \end{bmatrix}
\]

and a weight matrix, \( \mathbf{B}(n) \), for the recursive input vector, \( \mathbf{Y}_f(n) \), as

\[
\mathbf{B}(n) = \begin{bmatrix} \mathbf{B}_1(n) & \cdots & \mathbf{B}_M(n) \end{bmatrix}
\]

where

\[
\mathbf{B}_f(n) = \begin{bmatrix} b_{1(i-1)F+1}(n) & \cdots & b_{1(iF)}(n) \\ \vdots & \ddots & \vdots \\ b_{N(i-1)F+1}(n) & \cdots & b_{NiF}(n) \end{bmatrix}
\]

an adjustable weight matrix, \( \mathbf{W}(n) \), for the first layer of neurons can be written as

\[
\mathbf{W}(n) = \begin{bmatrix} w_{11}(n) & w_{12}(n) & \cdots & w_{1(L+1+MF)}(n) \\ \vdots & \ddots & \vdots \\ w_{N1}(n) & w_{N2}(n) & \cdots & w_{N(L+1+MF)}(n) \end{bmatrix} = [\mathbf{A}(n) \mid \mathbf{B}(n)].
\]

By defining the vector, \( \mathbf{S}(n) \), as

\[
\end{equation}
Defining the GRNN's performance is the commonly used mean square error (MSE) cost function minimum is called a gradient descent algorithm. In this paper, the cost function, \( s_i(n) \), of the first layer of neurons can be written as
\[
S(n) = \begin{bmatrix} s_1(n) \\ \vdots \\ s_N(n) \end{bmatrix} = W(n)U(n) = \begin{bmatrix} A(n) & B(n) \end{bmatrix} \begin{bmatrix} X(n) \\ Y_f(n) \end{bmatrix} = A(n)X(n) + B(n)Y_f(n), \tag{3}
\]

the output vector, \( O(n) \), of the first layer of neurons can be written as
\[
O(n) = \begin{bmatrix} q_1(n) \\ \vdots \\ q_N(n) \end{bmatrix} = f_1 \begin{bmatrix} s_1(n) \\ \vdots \\ s_N(n) \end{bmatrix} = f_1[S(n)]. \tag{4}
\]

The output vector, \( Y(n) \), of the network and the second layer of neurons can be written as
\[
Y(n) = \begin{bmatrix} f_2[q_1(n)] \\ \vdots \\ f_2[q_P(n)] \end{bmatrix} = f_2[Q(n)] = f_2[V(n)O(n)] \tag{5}
\]
where
\[
V(n) = \begin{bmatrix} v_{11}(n) & \cdots & v_{1N}(n) \\ \vdots & \ddots & \vdots \\ v_{P1}(n) & \cdots & v_{PN}(n) \end{bmatrix} = \begin{bmatrix} V_F(n) \\ V_O(n) \end{bmatrix}
\quad \text{and} \quad
Q(n) = \begin{bmatrix} q_1(n) \\ \vdots \\ q_P(n) \end{bmatrix} = \begin{bmatrix} Q_F(n) \\ Q_O(n) \end{bmatrix}.
\]

Substituting (3) into (4) and then (4) into (5), the network’s output can be written as
\[
Y(n) = f_2[V(n)f_1[W(n)U(n)]] \tag{6}
\]

To generate the error vector, \( E(n) \), the output vector, \( Y(n) \), is subtracted from the training vector, \( D(n) \), which is defined as
\[
D(n) = [d_1(n) \quad d_2(n) \quad \cdots \quad d_P(n)]^T,
\]

Thus, the error vector, \( E(n) \), is
\[
E(n) = [e_1(n) \quad e_2(n) \quad \cdots \quad e_P(n)]^T = D(n) - Y(n). \tag{7}
\]

3. A Real Time Recursive Backpropagation Algorithm for Two Layer GRNNs

A training, or optimization, algorithm that iteratively uses the gradient of the cost function or an estimate of the gradient of the cost function to determine a cost function’s minimum is called a gradient descent algorithm. In this paper, the cost function, \( J(n) \), defining the GRNN’s performance is the commonly used mean square error (MSE) cost function which implies that
\[
J(n) = E\left[ E^T(n)E(n) \right] = E \left[ \sum_{k=1}^{P} e_k^2(n) \right] = E \left[ \sum_{k=1}^{P} [d_k(n) - y_k(n)]^2 \right]. \tag{8}
\]
where \( E[\bullet] \) is the ensemble average function. Because an exact measurement of 
\( E[E^T(n)E(n)] \) is usually not available, an estimate, \( \hat{J}(n) \), of the cost function, \( J(n) \), is 
generally used to approximate the neural network’s performance. Backpropagation algo-
rithms are gradient descent algorithms that estimate the cost function, \( J(n) \), in (8) using 
only the current sample of \( J(n) \); that is, for LMS or backpropagation algorithms,

\[
J(n) = \hat{J}(n) = E^T(n)E(n) = \sum_{j=1}^{P} e_j^2(n) = \sum_{j=1}^{P} \left[ d_j(n) - y_j(n) \right]^2 . \tag{9}
\]

Backpropagation algorithms iteratively search the estimated cost function, \( \hat{J}(n) \), by ad-
justing the system’s coefficients so that the search for the cost function’s minimum begins 
from an initial point and proceeds in the direction opposite that of the gradient and in 
steps that are proportional to the gradient. In this section, a backpropagation algorithm 
that trains the two layer GRNN shown in Figure 1 by minimizing the estimated cost func-
tion, \( \hat{J}(n) \), in (8) and adjusting the elements in the matrices, \( W(n) \) and \( V(n) \), is de-
veloped.

To determine the optimal elements in the matrices, \( W(n) \) and \( V(n) \), the backpropa-
gation algorithm iteratively adjusts these matrices from their initial values in steps that 
are proportional to the negative of the gradient which implies that

\[
V(n+1) = V(n) - M(n) \frac{\partial \hat{J}(n)}{\partial V(n)} \tag{10}
\]

and

\[
W(n+1) = W(n) - M(n) \frac{\partial \hat{J}(n)}{\partial W(n)} \tag{11}
\]

where \( M(n) \) is the diagonal matrix,

\[
M(n) = \text{diag} \left[ \mu_0(n) \cdots \mu_L(n) \rho_1(n) \cdots \rho_{(MF)}(n) \right]
\]

where \( \mu_1(n), \mu_2(n), \ldots, \mu_L(n) \) and \( \rho_1(n), \rho_2(n), \ldots, \rho_{(MF)}(n) \) are the convergence factors for the nonrecursive inputs 
and \( \rho_{(MF)}(n) \) are the convergence factors for the (MF) recursive inputs. 
The convergence factors, \( \mu_0(n), \ldots, \mu_L(n), \rho_1(n), \ldots, \rho_{(MF)}(n) \), are positive values which can vary with each iteration. The elements in the matrices, \( W(0) = [A(0) \mid B(0)] \) 
and \( V(0) \), are initialized with small random numbers. The matrix, \( B(0) \), should be initial-
ized with values very near zero to ensure that the neural network is stable. Assuming that 
the algorithm does not converge to a local minimum, the steepest descent algorithm in 
(10) and (11) will have determined the optimal elements in the matrices, \( W(n) \) and 
\( V(n) \), when both gradients, \( \partial \hat{J}(n)/\partial W(n) \) and \( \partial \hat{J}(n)/\partial V(n) \), are zero. In practice, 
a neural network is considered to be trained when the cost function drops below a prede-
termined tolerance. Also, because backpropagation algorithms use the gradient of the 
estimated cost function, \( \hat{J}(n) \), the convergence factors in the matrix, \( M(n) \), are typically 
reduced as the gradients, \( \partial \hat{J}(n)/\partial W(n) \) and \( \partial \hat{J}(n)/\partial V(n) \), approach zero so as to reduce misadjustment.
Equations (9) and (10) represent the backpropagation algorithm for training the GRNN's second layer of neurons. In (10), the gradient of $\hat{J}(n)$ with respect to $V(n)$ can be written as
\[
\frac{\partial \hat{J}(n)}{\partial V(n)} = \begin{bmatrix} \frac{\partial \hat{J}(n)}{\partial v_{rc}(n)} \end{bmatrix}
\]
for $r = 1, 2, ..., P; c = 1, 2, ..., N$ where
\[
\hat{J}(n) = E^T(n)E(n)
\]
Substituting (7) into (12),
\[
\hat{J}(n) = E^T(n)E(n) = D^T(n)D(n) - 2D^T(n)Y(n) + Y^T(n)Y(n).
\]
Substituting (5) into (13),
\[
\hat{J}(n) = D^T(n)D(n) - 2D^T(n)Q[n] + Q[n]Q[n]T V(n)Y(n)
\]
The gradient of $\hat{J}(n)$ in (14) with respect to $V(n)$ is
\[
\frac{\partial \hat{J}(n)}{\partial V(n)} = -2\text{diag}[D(n)]\text{diag}\left[\frac{\partial^2 Q[n]}{\partial Q(n)}\right]O^T(n)
\]
\[
+ 2\text{diag}[Q[n]]\text{diag}[\frac{\partial^2 Q[n]}{\partial Q(n)}]O^T(n)
\]
\[
= -2\text{diag}[D(n)]\text{diag}[\frac{\partial^2 Q[n]}{\partial Q(n)}]O^T(n)
\]
\[
+ 2\text{diag}[Q[n]]\text{diag}[\frac{\partial^2 Q[n]}{\partial Q(n)}]O^T(n)
\]
\[
= -2\text{diag}[D(n) - Y(n)]\text{diag}[\frac{\partial^2 Q[n]}{\partial Q(n)}]O^T(n)
\]
\[
= -2\text{diag}[E(n)]\text{diag}[\frac{\partial^2 Q[n]}{\partial Q(n)}]O^T(n)
\]
where $\text{diag}[V]$ is a square diagonal matrix that has the elements of $V$ along its main diagonal if $V$ is a vector, and $\text{diag}[M]$ is a vector consisting of the diagonal elements of $M$ if $M$ is a square matrix. Thus, in (15),
\[
\text{diag}[E(n)] = \begin{bmatrix} e_1(n) & 0 & \cdots & 0 \\ 0 & e_1(n) & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & e_p(n) \end{bmatrix}^T
\]
and
\[
\text{diag}[\frac{\partial^2 Q[n]}{\partial Q(n)}] = \begin{bmatrix} \frac{\partial^2 Q_1[n]}{\partial q_1(n)} & \cdots & \frac{\partial^2 Q_p(n)}{\partial q_p(n)} \end{bmatrix}^T.
\]
Substituting (15) into (10), the backpropagation algorithm for training the second layer of neurons of the GRNN in Figure 1 can be written as
\[ V(n+1) = V(n) + 2M(n) \text{diag}[E(n)] \text{diag} \left[ \frac{\partial r}{\partial Q(n)} \right] O^T(n). \]  \hspace{1cm} (16)

Equation (16) is equivalent to the backpropagation algorithm used for training the second layer of neurons in a feedforward neural network with two layers of neurons.

Equations (9) and (11) represent the backpropagation algorithm for training the GRNN’s first layer of neurons. In (11), the gradient of \( \hat{J}(n) \) with respect to \( W(n) \) can be written as

\[
\frac{\partial \hat{J}(n)}{\partial W(n)} = \left[ \frac{\partial \hat{J}(n)}{\partial A(n)} \bigg| \frac{\partial \hat{J}(n)}{\partial B(n)} \right]
\]

Using the chain rule,

\[
\frac{\partial \hat{J}(n)}{\partial W(n)} = \left[ \frac{\partial \hat{J}(n)}{\partial w_{rc}(n)} \right] \sum_{i=1}^{N} \frac{\partial \hat{J}(n)}{\partial s_i(n)} \frac{\partial s_i(n)}{\partial w_{rc}(n)}
\]

for \( r = 1, 2, ..., N; c = 1, 2, ..., L+1+MF \),

\[
\frac{\partial \hat{J}(n)}{\partial A(n)} = \left[ \frac{\partial \hat{J}(n)}{\partial a_{rc}(n)} \right] \sum_{i=1}^{N} \frac{\partial \hat{J}(n)}{\partial s_i(n)} \frac{\partial s_i(n)}{\partial a_{rc}(n)}
\]

for \( r = 1, 2, ..., N; c = 0, 1, ..., L \) and

\[
\frac{\partial \hat{J}(n)}{\partial B(n)} = \left[ \frac{\partial \hat{J}(n)}{\partial b_{rc}(n)} \right] \sum_{i=1}^{N} \frac{\partial \hat{J}(n)}{\partial s_i(n)} \frac{\partial s_i(n)}{\partial b_{rc}(n)}
\]

for \( r = 1, 2, ..., N; c = 1, 2, ..., MF \). In matrix form, the gradient of \( \hat{J}(n) \) with respect to \( W(n) \) can be written as

\[
\frac{\partial \hat{J}(n)}{\partial W(n)} = \left[ \frac{\partial S(n)}{\partial A(n)} \bigg| \frac{\partial S(n)}{\partial B(n)} \right] \frac{\partial \hat{J}(n)}{\partial S(n)} = \left[ \alpha(n) \bigg| \beta(n) \right] \frac{\partial \hat{J}(n)}{\partial S(n)} \hspace{1cm} (17)
\]

where

\[
\alpha(n) = \left[ \alpha_{rcz}(n) \right] = \frac{\partial S(n)}{\partial A(n)} = \left[ \frac{\partial s_z(n)}{\partial a_{rc}} \right]
\]

for \( r = 1, 2, ..., N; c = 0, 1, ..., L; z = 1, 2, ..., N \) and

\[
\beta(n) = \left[ \beta_{rcz}(n) \right] = \frac{\partial S(n)}{\partial B(n)} = \left[ \frac{\partial s_z(n)}{\partial b_{rc}} \right]
\]

for \( r = 1, 2, ..., N; c = 1, 2, ..., MF; z = 1, 2, ..., N \). Substituting (17) into (11), the backpropagation algorithm for training the GRNN’s first layer of neurons can be written as

\[
W(n+1) = W(n) - M(n) \left[ \alpha(n) \bigg| \beta(n) \right] \frac{\partial \hat{J}(n)}{\partial S(n)} \hspace{1cm} (18)
\]

where

\[
\hat{J}(n) = E^T(n)E(n) = D^T(n)D(n) - 2D^T(n)Y(n) + Y^T(n)Y(n).
\]

By substituting (4) into (5) and (5) into (19), \( \hat{J}(n) \) can be written as
\[ \hat{J}(n) = D^T(n)D(n) - 2D^T(n)f_2 \left[ V(n)f_1 [S(n)] \right] + f_2^T \left[ V(n)f_1 [S(n)] \right] f_2 \left[ V(n)f_1 [S(n)] \right]. \] (20)

and the gradient of \( \hat{J}(n) \) in (20) with respect to \( S(n) \) is

\[
\frac{\partial \hat{J}(n)}{\partial S(n)} = -2 D_g \left[ \frac{\partial f_1 [S(n)]}{\partial S(n)} \right] V^T(n) \text{diag}[D(n)] \text{diag} \left[ \frac{\partial f_2 [Q(n)]}{\partial Q(n)} \right] \\
+ 2 D_g \left[ \frac{\partial f_1 [S(n)]}{\partial S(n)} \right] V^T(n) \text{diag}[Y(n)] \text{diag} \left[ \frac{\partial f_2 [Q(n)]}{\partial Q(n)} \right]
\]

\[
= -2 D_g \left[ \frac{\partial f_1 [S(n)]}{\partial S(n)} \right] V^T(n) \text{diag}[D(n) - Y(n)] \text{diag} \left[ \frac{\partial f_2 [Q(n)]}{\partial Q(n)} \right] \\
= -2 D_g \left[ \frac{\partial f_1 [S(n)]}{\partial S(n)} \right] V^T(n) \text{diag}[E(n)] \text{diag} \left[ \frac{\partial f_2 [Q(n)]}{\partial Q(n)} \right] 
\] (21)

where \( D_g[M] = \text{diag}[\text{diag}[M]] \). Thus, in (21),

\[ D_g \left[ \frac{\partial f_1 [S(n)]}{\partial S(n)} \right] = \text{diag} \left[ \frac{\partial f_1 [S_1(n)]}{\partial S_1(n)} \ldots \frac{\partial f_1 [S_N(n)]}{\partial S_N(n)} \right]^T. \]

Substituting (21) into (18), the backpropagation gradient descent algorithm for training the GRNN's second layer of neurons can be written as

\[ W(n + 1) = W(n) + 2M(n) \left[ \alpha(n) \mid \beta(n) \right] D_g \left[ \frac{\partial f_1 [S(n)]}{\partial S(n)} \right] \times V^T(n) \text{diag}[E(n)] \text{diag} \left[ \frac{\partial f_2 [Q(n)]}{\partial Q(n)} \right]. \] (22)

To calculate the matrix, \( \left[ \alpha(n) \mid \beta(n) \right] \), in (22), consider the terms, \( \alpha_{rcz}(n) = \frac{\partial s_z(n)}{\partial a_{rc}} \) and \( \beta_{rcz}(n) = \frac{\partial s_z(n)}{\partial b_{rc}} \), where

\[ s_z(n) = \sum_{k=0}^{L} a_{zk} x_k(n) + \sum_{i=1}^{M} \sum_{k=1}^{F} b_{z[i(i-1)F+k]} y_k(n-i), \] (23)

or equivalently,

\[ s_z(n) = \sum_{k=0}^{L} a_{zk} x_k(n) + \sum_{k=1}^{MF} b_{zk} y_k(n-i). \] (24)

Using (23),

\[ \alpha_{rcz}(n) = x_z(n) \delta(z-r) + \sum_{i=1}^{M} \sum_{k=1}^{F} b_{z[i(i-1)F+k]} \frac{\partial y_k(n-i)}{\partial a_{rc}}, \] (25)

and using (24),

\[ \beta_{rcz}(n) = y_z(n) \delta(z-r) + \sum_{k=1}^{MF} b_{zk} \frac{\partial y_k(n)}{\partial b_{rc}} \]

\[ = y_z(n) \delta(z-r) + \sum_{i=1}^{M} \sum_{k=1}^{F} b_{z[i(i-1)F+k]} \frac{\partial y_k(n-i)}{\partial b_{rc}}. \] (26)
In matrix form, (25) and (26) can be written as

$$\alpha(n) = [x_c(n)\delta(z-r)] + \frac{\partial Y_l(n)}{\partial A} B^T = [x_c(n)\delta(z-r)] + \sum_{i=1}^{M} \frac{\partial Y_F(n-i)}{\partial A} B_i^T$$

for \(r = 1, 2, ..., N; c = 0, 1, ..., L; z = 1, 2, ..., N\) and

$$\beta(n) = [y_i c(n)\delta(z-r)] + \frac{\partial Y_l(n)}{\partial B} B^T = [y_i c(n)\delta(z-r)] + \sum_{i=1}^{M} \frac{\partial Y_F(n-i)}{\partial B} B_i^T$$

\(r = 1, 2, ..., N; c = 1, 2, ..., MF; \) and \(z = 1, 2, ..., N\), respectively. To calculate \(\alpha_{rcz}(n)\) and \(\beta_{rcz}(n)\) using (25) and (26), expressions for \(\frac{\partial y_k(n)}{\partial a_{rc}}\) and \(\frac{\partial y_k(n)}{\partial b_{rc}}\) must be determined. From (2),

$$\frac{\partial y_k(n)}{\partial a_{rc}} = \frac{\partial f_2}{\partial q_k(n)} \sum_{l=1}^{N} v_{kl}(n) \frac{\partial f_1}{\partial s_l(n)} \frac{\partial s_l(n)}{\partial a_{rc}}$$

$$= \frac{\partial f_2}{\partial q_k(n)} \sum_{l=1}^{N} v_{kl}(n) \frac{\partial f_1}{\partial s_l(n)} \alpha_{rci}(n)$$

(27)

and

$$\frac{\partial y_k(n)}{\partial b_{rc}} = \frac{\partial f_2}{\partial q_k(n)} \sum_{l=1}^{N} v_{kl}(n) \frac{\partial f_1}{\partial s_l(n)} \frac{\partial s_l(n)}{\partial b_{rc}}$$

$$= \frac{\partial f_2}{\partial q_k(n)} \sum_{l=1}^{N} v_{kl}(n) \frac{\partial f_1}{\partial s_l(n)} \beta_{rci}(n).$$

(28)

Substituting (27) and (28) into (25) and (26), respectively,

$$\alpha_{rcz}(n) = x_c(n)\delta(z-r) + \sum_{i=1}^{M} \sum_{k=1}^{F} b_{z_l(i-1)F+k} \frac{\partial f_2}{\partial q_k(n-i)} \sum_{l=1}^{N} v_{kl}(n-i) \frac{\partial f_1}{\partial s_l(n-i)} \alpha_{rci}(n-i)$$

(29)

and

$$\beta_{rcz}(n) = y_i c(n)\delta(z-r) + \sum_{i=1}^{M} \sum_{k=1}^{F} b_{z_l(i-1)F+k} \frac{\partial f_2}{\partial q_k(n-i)} \sum_{l=1}^{N} v_{kl}(n-i) \frac{\partial f_1}{\partial s_l(n-i)} \beta_{rci}(n-i).$$

(30)

To express (29) and (30) in matrix form, define

$$\Phi(n) = D_g \left[ \frac{\partial f_1}{\partial s(n)} \right] V_F(n) D_g \left[ \frac{\partial f_2}{\partial Q_F(n)} \right]$$

The submatrix, \(\alpha(n)\), can then be expressed as
\[ \alpha(n) = \left[ x_c(n) \delta(z - r) \right] + \sum_{i=1}^{M} \alpha(n-i) \Phi(n-i) B_i^T(n) \]

for \( r = 1, 2, ..., N; \ c = 0, 1, ..., L; \ z = 1, 2, ..., N, \) and the submatrix, \( \beta(n) \), can be expressed as

\[ \beta(n) = \left[ y_{i,c}(n) \delta(z - r) \right] + \sum_{i=1}^{M} \beta(n-i) \Phi(n-i) B_i^T(n) . \]

for \( r = 1, 2, ..., N; \ c = 1, 2, ..., MF; \ z = 1, 2, ..., N. \)

Figure 2 shows a summary of the real time recursive backpropagation algorithm for training the GRNN shown in Figure 1.

4. Application of the RTRBP Algorithm

The algorithm derived in Section 3 is used to train two GRNNs to approximate two periodic multi-valued functions that feedforward neural networks are not capable of approximating. To illustrate why a FFNN cannot approximate many multi-valued periodic functions, consider a two layer FFNN which can uniformly approximate any continuous function on any compact set arbitrarily well regardless of the activation function in the first layer of neurons [1]. Because a FFNN has no feedback, it is memoryless; thus it can map an input or set of inputs to a single output or a single set of outputs, but it cannot map an input or set of inputs to multiple outputs or multiple sets of outputs. For example, consider approximating a function \( T \) that map \( x(n) \) into \( y(n) \); that is, consider approximating a function \( T \) such that \( y(n) = T[x(n)] \). If \( x(n) \) is periodic such that \( x(n) = x(n+50) \) and \( x(0) = x(50) = x(100) = 0 \) and the function \( T \) is defined such that \( y(n) = y(n+100) \) and \( y(0) = 1, \ y(50) = -1, \) and \( y(100) = 1, \)
then a FFNN that has the input \( x(n) = 0 \) can not determine if \( y(n) = 1 \) or \( y(n) = -1 \) without a past output. Thus, two periodic multi-valued functions are used to evaluate the RTRBP algorithm. The first function that the GRNN approximates is an analytic periodic multi-valued function, and the second function that the GRNN approximates is a nonanalytic periodic multi-valued function.

For the analytic periodic multi-valued function, the RTRBP algorithm trains a single nonrecursive input, single output GRNN that has 20 neurons in its first layer and uses the past 25 output samples, \( y(n-1), ..., y(n-25) \), as recursive inputs. The single nonrecursive input, \( x(n) \), is

\[ x(n) = \sin(0.04 \pi n) , \]

and the training signal, \( d(n) \), is

\[ d(n) = \cos(0.02 \pi n) . \]

Figure 3 shows one period of the GRNN’s output after training. The training signal is not shown because the two curves appear coincident.
Step 1. Initialize $W(0)$, $V(0)$, $\alpha(0)$, $\beta(0)$ and $n$.

1.1 Initialize $V(0)$ and $W(0) = [A(0) \mid B(0)]$ with small random numbers.

1.1.1 The matrix, $B(0)$, should be initialized to ensure that the neural network is stable.

1.2 Initialize $\alpha(0) = 0$ and $\beta(0) = 0$ unless the values of $\alpha(0)$ or $\beta(0)$ are known.

1.3 Set $n = 0$.

Step 2 While training the GRNN, do Steps 3-7.

Step 3 Calculate the GRNN’s response, $Y(n)$, to the input, $U(n)$.

3.1 $S(n) = W(n)U(n)$.

3.2 $O(n) = f_1[S(n)]$.

3.3 $Q(n) = V(n)O(n)$.

3.3.1 Extract $Q_F(n)$ from the first $F$ rows of $Q(n)$.

3.4 $Y(n) = f_2[Q(n)]$.

3.4.1 Extract $Y_F(n)$ from the first $F$ rows of $Y(n)$.

Step 4 Calculate the training error vector, $E(n)$.

4.1 $E(n) = D(n) - Y(n)$

Step 5 Calculate the following derivatives.

5.1 $Dg\left[\frac{\partial f_1[S(n)]}{\partial S(n)}\right] = \text{diag}\left[\text{diag}\left[\frac{\partial f_1[S(n)]}{\partial S(n)}\right]\right]$

5.2 $Dg\left[\frac{\partial f_2[Q(n)]}{\partial Q(n)}\right]$

5.2.1 Extract $Dg\left[\frac{\partial f_2[Q(n)]}{\partial Q(n)}\right]$ from the first $F$ rows of $Dg\left[\frac{\partial f_2[Q(n)]}{\partial Q(n)}\right]$

Step 6 Calculate $\Phi(n)$, $\alpha(n)$ and $\beta(n)$ and save results for $M$ training iterations.

6.1 $\Phi(n) = Dg\left[\frac{\partial f_1[S(n)]}{\partial S(n)}\right]V_F^T(n)Dg\left[\frac{\partial f_2[Q_F(n)]}{\partial Q_F(n)}\right]$

6.2 $\alpha(n) = \left[x_c(n)\delta(z - r)\right] + \sum_{i=1}^{M}\alpha(n - i)\Phi(n - i)B_F^T(n)$

$r = 1, 2, \ldots, N; c = 0, 1, \ldots, L; z = 1, 2, \ldots, N.$

6.3 $\beta(n) = \left[y_c(n)\delta(z - r)\right] + \sum_{i=1}^{M}\beta(n - i)\Phi(n - i)B_F^T(n)$

$r = 1, 2, \ldots, N; c = 1, 2, \ldots, MF; z = 1, 2, \ldots, N.$

Step 7 Adjust elements in the matrices, $W(n)$ and $V(n)$, and update $n$.

7.1 $W(n + 1) = W(n) + 2M(n)\left[\alpha(n) \mid \beta(n)\right]$

$\times Dg\left[\frac{\partial f_1[S(n)]}{\partial S(n)}\right]V^T(n)\text{diag}[E(n)]\text{diag}\left[\frac{\partial f_2[Q(n)]}{\partial Q(n)}\right]$

7.2 $V(n + 1) = V(n) + 2M(n)\text{diag}[E(n)]\text{diag}\left[\frac{\partial f_2[Q(n)]}{\partial Q(n)}\right]O^T(n)$

7.3 $n = n + 1$

Figure 2. Summary of the Real Time Recursive Backpropagation Algorithm for the two layer GRNN shown in Figure 1.
For the nonanalytic periodic multi-valued function, the algorithm again trains a single nonrecursive input, single output GRNN that has 20 neurons in its first layer and uses the past 25 output samples, \( y(n-1), \ldots, y(n-25) \), as recursive inputs. However, for this example the single nonrecursive input, \( x(n) \), is the nonanalytic periodic triangle wave, 
\[
x(n) = \arcsin[\sin(0.08\pi n)]
\]
and the training signal, \( d(n) \), is the triangle wave, 
\[
d(n) = \arcsin[\sin(0.02\pi n - \pi/2)]
\]
Figure 4 shows the training signal as the dashed line and one period of the GRNN’s output as the solid line.

5. Summary

In this paper, a two layer globally recursive neural network (GRNN) that has multiple delays in its feedback path is described mathematically using a simple vector matrix description. A real time backpropagation algorithm, referred to as a real time recursive backpropagation (RTRBP) algorithm, is developed for this two layer GRNN, and is also described using a set of simple vector matrix equations. This RTRBP algorithm is a gradient descent algorithm that calculates the gradient using a set of recursive vector matrix
equations that are of the same order as the order of the GRNN; that is, the order of the set of recursive vector matrix equations that calculate the gradient is the same as the number of delays in the GRNN’s feedback path. This RTRBP algorithm is applied to two GRNNs that are trained to approximate two periodic multi-valued functions. These periodic multi-valued functions were chosen because FFNNs are memoryless and therefore cannot approximate multi-valued functions. As a result, FFNNs cannot model systems with hysteresis, because systems with hysteresis are modeled with multi-valued functions. The first function that the GRNN approximates is an analytic periodic multi-valued function, and the second function that the GRNN approximates is a nonanalytic periodic multi-valued function. The GRNN approximates the analytic function much better than the nonanalytic function.

The performance of gradient descent algorithms, such as the RTRBP algorithm developed in this paper, are a function the algorithm’s convergence factors. Convergence factors can affect a gradient descent algorithm’s stability, learning rate, and misadjustment. Future research will examine the relationship between the RTRBP algorithm’s convergence factors and its stability, learning rate, and misadjustment.

**Figure 4.** Output of a GRNN that has the single nonrecursive input, \(x(n)\), where \(x(n) = \text{arcsin} \left( \sin(0.08\pi n) \right)\), and the training signal, \(d(n)\), where \(d(n) = \text{arcsin} \left( \sin(0.02\pi n - \pi/2) \right)\).
6. References


