Express Letters

Complex Domain Backpropagation

George M. Georgiou and Cris Koutsougeras

Abstract—**The well-known backpropagation algorithm is extended to complex domain backpropagation (CDBP) which can be used to train neural networks for which the inputs, weights, activation functions, and outputs are complex-valued. Previous derivations of CDBP were necessarily admitting activation functions that have singularities, which is highly undesirable. Here CDBP is derived so that it accommodates classes of suitable activation functions. One such function is found and the circuit implementation of the corresponding neuron is given. CDBP hardware circuits can be used to process sinusoidal signals all at the same frequency (phasors).**

I. Introduction

The complex domain backpropagation (CDBP) algorithm provides a means for training feed-forward networks for which the inputs, weights, activation functions, and outputs are complex-valued. CDBP, being nonlinear in nature, is more powerful than the complex LMS algorithm [1] in the same way that usual (real) backpropagation [2] is more powerful than the (real) LMS algorithm [3]. Thus, CDBP can replace the complex LMS algorithm in applications such as filtering in the frequency domain [4].

In addition, CDBP can be used to design networks in hardware that process sinusoidal inputs all having the same frequency (phasors). Such signals are commonly represented by complex values. The complex weights of the neural network represent impedance as opposed to resistance in real backpropagation networks. The desired outputs are either sinusoids at the same frequency as the inputs or, after further processing, binary.

By decomposing the complex numbers that represent the inputs and the desired outputs into real and imaginary parts, the usual (real) backpropagation algorithm [2] can be used to train a conventional neural network which performs, in theory, the same input–output mapping without involving complex numbers. However, such networks, when implemented in hardware, cannot accept the sinusoidal inputs (phasors) and give the desired outputs.

In [5] and [7][1] complex domain backpropagation algorithms were independently derived. The derivation was single hidden layer feed-forward network in [5] and for a multilayer network in [6]. Both derivations are based on the assumption that the (complex) derivative $f'(z) = df(z)/dz$ of the activation function $f(z)$ exists. Here we will show that if the derivative $f'(z)$ exists for all $z \in \mathbb{C}$, the complex plane, then $f(z)$ is not an appropriate activation function. We will derive CDBP for multilayer feed-forward networks imposing less stringent conditions on $f(z)$ so that more appropxiate activation functions can be used. One such function is found and its circuit implementation is given.

Manuscript received September 9, 1991; revised February 19, 1992. This paper was recommended by Associate Editor R. Liu.

The authors are with the Computer Science Department, Tulane University, New Orleans, LA 70118.

IEEE Log Number 9200589.

[1] This paper [6] was published at about the same time that this letter was submitted. It came to our attention soon after submission.

More information on the definitions and results from the theory of complex variables that we will use can be found in [7], [8], or in any other introductory book on the subject.

II. Complex Domain Backpropagation

Let the neuron activation function be $f(z) = u(x, y) + iv(x, y)$ where $z = x + iy$, $i = \sqrt{-1}$. The functions u and v are the real and imaginary parts of f, respectively. Likewise, x and y are the real and imaginary parts of z. For now it is sufficient to assume that the partial derivatives $u_x = \partial u/\partial x$, $u_y = \partial u/\partial y$, $v_x = \partial v/\partial x$, and $v_y = \partial v/\partial y$ exist for all $z \in \mathbb{C}$. This assumption is not sufficient that $f'(z)$ exists. If it exists, the partial derivatives also have to satisfy the Cauchy–Riemann equations (see next section). The error for an input pattern is defined as

$$E = \frac{1}{2}\sum_k \epsilon_k \bar{\epsilon}_k, \qquad \epsilon_k = d_k - o_k \tag{1}$$

where d_k is the desired output and o_k the actual output of the kth output neuron. The overbar signifies complex conjugate. It should be noted that the error E is a real scalar function. The output o_j of neuron j in the network is

$$o_j = f(z_j) = u^j + iv^j, \qquad z_j = x_j + iy_j = \sum_{l=1} W_{jl}X_{jl} \tag{2}$$

where the W_{jl}'s are the (complex) weights of neuron j and X_{jl}'s their corresponding (complex) inputs. A bias weight, having permanent input (1,0), may also be added to each neuron. In what follows, the subscripts R and I indicate the real and imaginary parts, respectively. We note the following partial derivatives:

$$\frac{\partial x_j}{\partial W_{jlR}} = X_{jlR}, \quad \frac{\partial y_j}{\partial W_{jlR}} = X_{jlI}, \quad \frac{\partial x_j}{\partial W_{jlI}} = -X_{jlI}, \quad \frac{\partial y_j}{\partial W_{jlI}} = X_{jlR}. \tag{3}$$

In order to use the chain rule to find the gradient of the error function E with respect to the real part of W_{jl}, we have to observe the variable dependencies: The real function E is a function of both $u^j(x_j, y_j)$ and $v^j(x_j, y_j)$, and x_j and y_j are both functions of W_{jlR} (and W_{jlI}). Thus, the gradient of the error function with respect to the real part of W_{jl} can be written as

$$\frac{\partial E}{\partial W_{jlR}} = \frac{\partial E}{\partial u^j}\left(\frac{\partial u^j}{\partial x_j}\frac{\partial x_j}{\partial W_{jlR}} + \frac{\partial u^j}{\partial y_j}\frac{\partial y_j}{\partial W_{jlR}}\right) + \frac{\partial E}{\partial v^j}\left(\frac{\partial v^j}{\partial x_j}\frac{\partial x_j}{\partial W_{jlR}} + \frac{\partial v^j}{\partial y_j}\frac{\partial y_j}{\partial W_{jlR}}\right)$$

$$= -\delta_{jR}\left(u_x^j X_{jlR} + u_y^j X_{jlI}\right) - \delta_{jI}\left(v_x^j X_{jlR} + v_y^j X_{jlI}\right) \quad (4)$$

with $\delta_j \overset{\text{def}}{=} -\partial E/\partial u^j - i\partial E/\partial v^j$ and consequently $\delta_{jR} = -\partial E/\partial u^j$ and $\delta_{jI} = -\partial E/\partial v^j$. Likewise, the gradient of the error function with respect to the imaginary part of W_{jl} is

$$\frac{\partial E}{\partial W_{jlI}} = \frac{\partial E}{\partial u^j}\left(\frac{\partial u^j}{\partial x_j}\frac{\partial x_j}{\partial W_{jlI}} + \frac{\partial u^j}{\partial y_j}\frac{\partial y_j}{\partial W_{jlI}}\right) + \frac{\partial E}{\partial v^j}\left(\frac{\partial v^j}{\partial x_j}\frac{\partial x_j}{\partial W_{jlI}} + \frac{\partial v^j}{\partial y_j}\frac{\partial y_j}{\partial W_{jlI}}\right)$$
$$= -\delta_{jR}\left(u_x^j(-X_{jlI}) + u_y^j X_{jlR}\right) - \delta_{jI}\left(v_x^j(-X_{jlI}) + v_y^j X_{jlR}\right). \quad (5)$$

Combining (4) and (5), we can write the gradient of the error function E with respect to the complex weight W_{jl} as

$$\nabla_{W_{jl}} E \overset{\text{def}}{=} \frac{\partial E}{\partial W_{jlR}} + i\frac{\partial E}{\partial W_{jlI}}$$
$$= -\overline{X}_{jl}\left(\left(u_x^j + iu_y^j\right)\delta_{jR} + \left(v_x^j + iv_y^j\right)\delta_{jI}\right). \quad (6)$$

To minimize the error E, each complex weight W_{jl} should be changed by a quantiy ΔW_{jl} proportional to the negative gradient:

$$\Delta W_{jl} = \alpha\overline{X}_{jl}\left(\left(u_x^j + iu_y^j\right)\delta_{jR} + \left(v_x^j + iv_y^j\right)\delta_{jI}\right) \quad (7)$$

where α is the learning rate, a real positive constant. A momentum term [2] may be added in the above learning equation. When the weight W_{jl} belongs to an output neuron, then δ_{jR} and δ_{jl} in (6) have the values

$$\delta_{jR} = -\frac{\partial E}{\partial u^j} = \epsilon_{jR} = d_{jR} - u^j \text{ and } \delta_{jI} = -\frac{\partial E}{\partial v^j} = \epsilon_{jI} = d_{jI} - v^j \quad (8)$$

or more compactly:

$$\delta_j = \epsilon_j = d_j - o_j. \quad (9)$$

When weight W_{jl} belongs to a hidden neuron, i.e., when the output of the neuron is fed to other neurons in subsequent layers, in order to compute δ_j, or equivalently δ_{jR} and δ_{jI}, we have to use the chain rule. Let the index k indicate a neuron that receives input from neuron j. Then the net input z_k to neuron k is

$$z_k = x_k + iy_k = \sum_l \left(u^l + iv^l\right)\left(W_{klR} + iW_{klI}\right) \quad (10)$$

where the index l runs through the neurons from which neuron k receives input. Thus, we have the following partial derivatives:

$$\frac{\partial x_k}{\partial u^j} = W_{kjR}, \frac{\partial y_k}{\partial u^j} = W_{kjI},$$
$$\frac{\partial x_k}{\partial v^j} = -W_{kjI}, \frac{\partial y_k}{\partial v^j} = W_{kjR}. \quad (11)$$

Using the chain rule we compute δ_{jR}:

$$\delta_{jR} = -\frac{\partial E}{\partial u^j} = -\sum_k \frac{\partial E}{\partial u^k}\left(\frac{\partial u^k}{\partial x_k}\frac{\partial x_k}{\partial u^j} + \frac{\partial u^k}{\partial y_k}\frac{\partial y_k}{\partial u^j}\right) - \sum_k \frac{\partial E}{\partial v^k}\left(\frac{\partial v^k}{\partial x_k}\frac{\partial x_k}{\partial u^j} + \frac{\partial v^k}{\partial y_k}\frac{\partial y_k}{\partial u^j}\right)$$
$$= \sum_k \delta_{kR}\left(u_x^k W_{kjR} + u_y^k W_{kjI}\right) + \sum_k \delta_{kI}\left(v_x^k W_{kjR} + v_y^k W_{kjI}\right) \quad (12)$$

where the index k runs through the neurons that receive input from neuron j. In a similar manner we can compute δ_{jI}:

$$\delta_{jI} = -\frac{\partial E}{\partial v^j} = -\sum_k \frac{\partial E}{\partial u^k}\left(\frac{\partial u^k}{\partial x_k}\frac{\partial x_k}{\partial v^j} + \frac{\partial u^k}{\partial y_k}\frac{\partial y_k}{\partial v^j}\right) - \sum_k \frac{\partial E}{\partial v^k}\left(\frac{\partial v^k}{\partial x_k}\frac{\partial x_k}{\partial v^j} + \frac{\partial v^k}{\partial y_k}\frac{\partial y_k}{\partial v^j}\right)$$
$$= \sum_k \delta_{kR}\left(u_x^k(-W_{kjI}) + u_y^k W_{kjR}\right) + \sum_k \delta_{kI}\left(v_x^k(-W_{kjI}) + v_y^k W_{kjR}\right) \quad (13)$$

Combining (12) and (13), we arrive at the following expression for δ_j:

$$\delta_j = \delta_{jR} + i\delta_{jI} = \sum_k \overline{W}_{kj}\left(\left(u_x^k + iu_y^k\right)\delta_{kR} + \left(v_x^k + iv_y^k\right)\delta_{kI}\right). \quad (14)$$

Training of a feed-forward network with the CDBP algorithm is done in a similar manner as in the usual (real) backpropagation [2]. First, the weights are initialized to small random complex values. Until an acceptable output error level is arrived at, each input vector is presented to the network, the corresponding output and output error are calculated (forward pass), and then the error is backpropagated to each neuron in the network and the weights are adjusted accordingly (backward pass). More precisely, for input pattern X_j, δ_j for neuron j is computed by starting at the neurons in the output layer using (9) and then for neurons in hidden layers by recursively using (14). As soon as δ_j is computed for neuron j, its weights are changed according to (7).

III. The Activation Function

In the next section the important properties that a suitable activation function $f(z)$ must possess will be discussed. For the purposes of this section, we identify two properties that a suitable $f(z)$ should possess: i) $f(z)$ is bounded (see next section), and ii) $f(z)$ is such that $\delta_j \neq (0,0)$ and $X_{jl} \neq (0,0)$ imply $\nabla_{W_{jl}} E \neq (0,0)$. Noncompliance with the latter condition is undesirable since it would imply that even in the presence of both non-zero input ($X_{jl} \neq (0,0)$) and non-zero error ($\delta_j \neq (0,0)$) it is still possible that $\Delta W_{jl} = 0$, i.e., no learning takes place.

The deviations of CDBP in [5] and [6] are based on the assumption that the (complex) derivative $f'(z)$ of the activation function exists without giving a specific domain. If the domain is taken to be $\mathbb{C}$, then such functions are called *entire*.

Definition 1: A function $f(z)$ is called *analytic* at a point z_0 if its derivative exists throughout some neighborhood of z_0. If $f(z)$ is analytic at all points $z \in \mathbb{C}$, is called *entire.*

Proposition 1: Entire functions are not suitable activation functions.

The proof of the proposition follows directly from Liouville's theorem [7], [8].

Theorem 1 (Liouville): If $f(z)$ is entire and bounded on the complex plane, then $f(z)$ is a constant function.

Since a suitable $f(z)$ must be bounded, it follows from Liouville's theorem that if in addition $f(z)$ is entire, then $f(z)$ is constant—clearly not a suitable activation function. In fact, the requirement that $f(z)$ be entire alone imposes considerable structure on it, e.g., the Cauchy–Riemann equations should be satisfied. An exception in which $f(z)$ is taken to be entire is the complex LMS algorithm (single neuron) [1], where $f(z) = z$, the identity function.

Proposition 2: If $f(z)$ is entire, then (6) becomes

$$\nabla_{W_{jl}} E = -\overline{X}_{jl} \delta_j \overline{f'}(z_j) \tag{15}$$

and (14) becomes

$$\delta_j = \sum_k \overline{W}_{kj} \delta_k \overline{f'}(z_k). \tag{16}$$

Proof: Since $f(z_j)$ is entire, the Cauchy–Riemann equations $u_x^j = v_y^j$ and $u_y^j = -v_x^j$ are satisfied and the derivative of $f(z_j)$ is $f'(z_j) = u_x^j + iv_x^j$. Using these equations we can write (6) as

$$\begin{aligned}\nabla_{W_{jl}} E &= -\overline{X}_{jl}\left(\left(u_x^j - iv_x^j\right)\delta_{jR} + \left(v_x^j + iu_x^j\right)\delta_{jI}\right)\\ &= -\overline{X}_{jl}\delta_j \overline{f'}(z_j).\end{aligned} \tag{17}$$

In a similar way we may derive (16). Expressions (15) and (16) are essentially the ones derived in [5] and [6]. If the activation function is the identity function, $f(z) = z$, and if neuron j is an output neuron, then $f'(z) = 1$ and the gradient (15) further reduces to the one in the complex LMS (delta) rule [1]:

$$\nabla_{W_{jl}} E = -\overline{X}_{jl} \epsilon_j. \tag{18}$$

Proposition 3: If $u_x v_y \equiv v_x u_y$, then $f(z) = u + iv$ is not a suitable activation function.

Proof: We would like to show that an activation function with the above property violates condition ii) (see beginning of section). Suppose that $X_{jl} \neq (0, 0)$. We will show that there exists $\delta_j \neq (0, 0)$ such that $\nabla_{W_{jl}} E = 0$. From (6) we see that $\nabla_{W_{jl}} E = 0$ when $(u_x^j \delta_{jR} + v_x^j \delta_{jI}) + i(u_y^j \delta_{jR} + v_y^j \delta_{jI}) = 0$, or equivalently, when the real and imaginary parts of the equation equal to zero:

$$\begin{aligned} u_x^j \delta_{jR} + v_x^j \delta_{jI} &= 0\\ u_y^j \delta_{jR} + v_y^j \delta_{jI} &= 0. \end{aligned} \tag{19}$$

A nontrivial solution of the above homogeneous system of equations in δ_{jR} and δ_{jI}, i.e., δ_{jR} and δ_{jI} not both zero, can be found if and only if the determinant of the coefficient matrix is zero: $u_x^j v_y^j - v_x^j u_y^j = 0$, or $u_x^j v_y^j = v_x^j u_y^j$.

Corollary: If $u \equiv v + k$, where k is any complex constant, then $f(z) = u + iv$ is not a suitable activation function.

The corollary gives us a better feeling of how large the class of unsuitable activation functions really is.

IV. A Suitable Activation Function

In [5] the sigmoid function $g(z) = 1/(1 + e^{-z})$ was incorrectly assumed to remain bounded when its domain was extended from $\mathbb{R}$ to $\mathbb{C}$, i.e. there exists a real number M such that $|g(z)| \leq M$ for all $z \in \mathbb{C}$. It can be easily verified that when z approaches any value in the set $\{0 \pm i(2n + 1)\pi : n \text{ is any integer}\}$, then $|g(z)| \to \infty$, and thus $g(z)$ is unbounded. Furthermore, other commonly used activation functions, e.g., $\tanh(z)$ and e^{-z^2} [10], are unbounded as well when their domain is extended from the real line to the complex plane. One can verify that $|\tanh(z)| \to \infty$ as z approaches a value in the set $\{0 \pm i((2n + 1)/2)\pi : n \text{ is any integer}\}$ and $|e^{-z^2}| \to \infty$ when $z = 0 + iy$ and $y \to \infty$.

In order to avoid the problem of singularities in the sigmoid function $g(z)$, it was suggested in [6] to scale "the input data to some region in the complex plane." Backpropagation being a weak optimization procedure has no mechanism of constraining the values the weights can assume, and thus the value of z, which depends on both the inputs and the weights, can take any value on the complex plane. Based on this observation the suggested remedy is inadequate.

It is clear that some other activation function must be found for CDBP. In the derivation of CDBP we only assumed that the partial derivatives u_x, u_y, v_x, and v_y exist. Other important properties the activation function $f(z) = u(x, y) + iv(x, y)$ should possess are the following

- $f(z)$ is nonlinear in x and y. If the function is linear, then the capabilities of the network are severely limited, i.e., it is impossible to train a feed-forward neural network to perform a nonlinearly separable classification problem.
- $f(z)$ is bounded. This is true if and only if both u and v bounded. Since both u and v are used during training (forward pass), they must be bounded. If either one was unbounded, then a software overflow could occur. One could image analogous complications if the algorithm is implemented in hardware.
- The partial derivatives u_x, u_y, v_x, and v_y exist and are bounded. Since all are used during training, they must be bounded.
- $f(z)$ is not entire. See Proposition 1 in the previous section.
- $u_x v_y \not\equiv v_x u_y$. See Proposition 3 in previous section.

A simple function that satisfies all the above properties is

$$f(z) = \frac{z}{c + \frac{1}{r}|z|} \tag{20}$$

where c and r are real positive constants. This function has the property of mapping a point $z = x + iy = (x, y)$ on the complex plane to a unique point $f(z) = (x/(c + (1/r)|z|), y/(c + (1/r)|z|))$ on the open disc $\{z : |z| < r\}$. The phase angle of z is the same as that of $f(z)$. The magnitude $|z|$ is monotonically squashed to $|f(z)|$, a point in the interval $[0, r)$; in an analogous way, the real sigmoid and hyperbolic tangent functions map a point x on the real line to a unique point on a finite interval. Parameter c controls the steepness of $|f(z)|$. The partial derivatives u_x, u_y, v_x, and v_y are

$$u_x = \begin{cases} \dfrac{r(y^2 + cr|z|)}{|z|(cr + |z|)^2} & \text{if } |z| \neq 0 \\ \dfrac{1}{c} & \text{if } |z| = 0 \end{cases}$$

$$u_y = \begin{cases} -\dfrac{rxy}{|z|(cr + |z|)^2} & \text{if } |z| \neq 0 \\ 0 & \text{if } |z| = 0 \end{cases}$$

TABLE I

	Input Pattern (and Desired Output)	Actual Outputs After 1500 Epochs
1	$\{(1, 0), (0, 0), (0, 0)\}$	$\{(0.9166E + 00, 0.4907E - 03), (0.4681E - 01, 0.2388E - 02), (-0.3326E - 02, 0.4553E - 01)\}$
2	$\{(0,, 0), (1, 0), (0, 0)\}$	$\{(-0.2776E - 01, -0.2885E - 01), (0.9136E + 00, -0.7155E - 03), (0.2145E - 01, -0.2797E - 01)\}$
3	$\{(0, 0), (0, 0), (0, -1)\}$	$\{(0.3028E - 01, -0.3061E - 01), (-0.3409E - 01, 0.1932E - 01), (0.1474E - 02, -0.9148E + 00)\}$

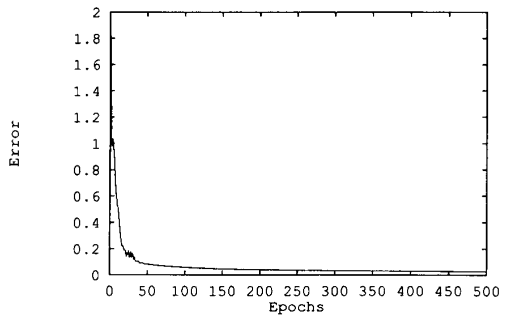

Fig. 1. A run of CDBP.

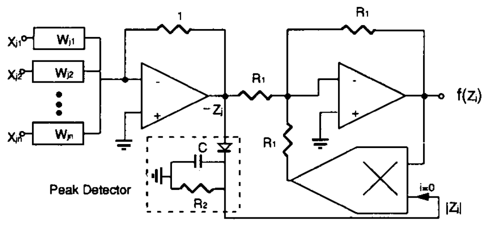

Fig. 2. Circuit for "complex" neuron. The output voltage is $f(z_j) = z_j/(1 + |z_j|)$, *where* $z_j = \sum_{i=1}^{n} X_{ji}W_{ji}$.

$$v_x = \begin{cases} -\dfrac{rxy}{|z|(cr + |z|)^2} & \text{if } |z| \neq 0 \\ 0 & \text{if } |z| = 0 \end{cases}$$

$$v_y = \begin{cases} \dfrac{r(x^2 + cr|z|)}{|z|(cr + |z|)^2} & \text{if } |z| \neq 0 \\ \dfrac{1}{c} & \text{if } |z| = 0. \end{cases} \tag{21}$$

The special definitions of the partial derivatives when $|z| = 0$, i.e., at $z = (0, 0)$, correspond to their limits as $z \to (0, 0)$. Being thus defined, the singularities at the origin are removed and all partial derivatives exist and are continuous for all $z \in \mathbb{C}$.

We successfully tested the CDBP algorithm on a number of training sets. The error-versus-epochs graph of a simulation example appears in Fig. 1. By "error" we mean the sum of the errors due to each pattern in the training set (1). A form of the encoding problem [2] was used: The desired output of each of the three input patterns in Table I is the input pattern itself. We used a 3-2-3 network with $c = 1$, $r = 1$, $\alpha = 0.2$, and momentum rate 0.9. The actual outputs after 1500 epochs appear in Table I.

Since sinusoidal signals at the same frequency (phasors) are commonly represented and manipulated as complex numbers, it is natural to use CDBP for processing them. CDBP-trained neural networks with activation function (20) can be directly translated to analog circuits. Such circuits, though nonlinear, yield outputs at the same frequency (see below).

A simple circuit diagram for the "complex" neuron j with activation function $f(z)$ when $c = 1$ and $r = 1$ appears in Fig. 2. The inputs and output are voltages. The quantity $|z_j|$ represents the amplitude of the sinusoidal voltage z_j and in the circuit is obtained as a dc voltage using a peak detector (demodulator) with input $-z_j$. If the ripple associated with the output of the peak detector is unacceptable, then the peak detector should be followed by a low-pass filter [10]. Each weight W_{ji} represents admittance.

Proposition 4: If all input signals to a neural network circuit consisting of neurons of the type that appears in Fig. 2 are sinusoids at the same frequency ω, then the output(s) is (are) sinusoid(s) at the same frequency ω.

Proof: It is sufficient to consider a single neuron (Fig. 2). Given that the X_{ji}'s are sinusoids at the frequency ω, certainly the output $-z_j$ of the summer (first op-amp) is a sinusoid at frequency ω. The remaining circuitry inverts $-z_j$ and scales its *magnitude* by a factor $1/(c + (1/r)|z_j|)$; thus, there is no effect on the frequency and the output $f(z_j)$ is also a sinusoid at frequency ω.

It is interesting to note that the activation function $f(z)$, $z \in \mathbb{R}$, being a sigmoid, can be used in the usual (real) backpropagation [11], [12]. In that case the neuron circuit of Fig. 2 can also be used, with the exception that the peak detector is replaced with a full-wave rectifier in order to obtain the absolute value $|z|$ from the now real $-z$; weights W_{ji} represent conductance.

V. Conclusion

CDBP is a generalization of the usual (real) backpropagation to handle complex-valued inputs, weights, activation functions, and outputs. Unlike previous attempts, the CDBP algorithm was derived so that it can accommodate suitable activation functions, which were discussed. A simple suitable activation function was found and the circuit implementation of the corresponding "complex" neuron was given. The circuit can be used to process sinusoids at the same frequency (phasors).

References

[1] B. Widrow, J. McCool, and M. Ball, "The complex LMS algorithm," *Proc. IEEE,* vol. 63, pp. 719-720, 1975.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Volume 1: Foundations,* D. E. Rumelhart and J. L. McClelland, Eds. Cambridge: MIT Press, 1986, pp. 318-362.

[3] B. Widrow and M. Hoff, "Adaptive switching circuits," in *1960 IRE*

WESCON Convention Record, part 4. New York: IRE, 1960, pp. 96-104.
[4] M. Dentino, J. McCool, and B. Widrow, "Adaptive filtering in the frequency domain," *Proc. IEEE,* vol. 66, pp. 1658-1659, Dec. 1978.
[5] M. S. Kim and C. C. Guest, "Modification of backpropagation for complex-valued signal processing in frequency domain," in *IJCNN Int. Joint Conf. Neural Networks,* pp. III-27-III-31, June 1990.
[6] H. Leung and S. Haykin, "The complex backpropagation algorithm," *IEEE Trans. Signal Processing,* vol. 39, pp. 2101-2104, Sept. 1991.
[7] R. V. Churchill and J. W. Brown, *Complex Variables and Applications.* New York: McGraw-Hill, 1948.
[8] F. P. Greenleaf, *Introduction of Complex Variables.* W. B. Saunders, 1972.
[9] B. Kosko, *Neural Networks and Fuzzy Systems.* Englewood Cliffs: NJ, Prentice-Hall, 1992.
[10] J. Millman, *Microelectronics.* New York: McGraw-Hill, 1979.
[11] P. Thrift, "Neural networks and nonlinear modeling," *Texas Instruments Tech. J.,* vol. 7, Nov.-Dec. 1990.
[12] E. B. Stockwell, "A fast activation function for backpropagation networks," *Tech. Rep.,* Astronomy, Univ. Minnesota, May 1991.

Reply to "Comments on 'Pole and Zero Estimation in Linear Circuits'"

Stephen B. Haley and Paul J. Hurst

This reply appears on p. 419 of the May 1992 issue of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: FUNDAMENTAL THEORY AND APPLICATIONS.

IEEE Log Number 9200418.