The Field of Values of a Matrix and Neural Networks

George M. Georgiou, *Senior Member, IEEE*

***Abstract*—The field of values of a matrix, also known as the numerical range, is introduced in the context of neural networks. Using neural network techniques, an algorithm and a generalization are developed that find eigenpairs of a normal matrix. The dynamics of the algorithm can be observed on the complex plane. Only limited visualization is possible in the case when the matrix is Hermitian (or real symmetric) since the field of values is confined on the real line. The eigenpairs can serve as stored memories, which are recalled by using the algorithm. Shifting in the algorithm is also discussed, which assists in finding other eigenpairs. Trajectories of runs of the algorithm are visually presented, through which the behavior of the algorithms is elucidated.**

***Index Terms*—Complex-valued neural networks, field of values, numerical range, eigenvectors, eigenvalues, normal matrices.**

I. Introduction

In this paper, the *field of values* of a matrix [1] [2], which is also known as the *numerical range*, is introduced in the context of neural networks, and in particular, in that of complex-valued neural networks. It seems that the field of values has not been previously used in any significant way in neural networks or in other engineering applications. For an $n \times n$ matrix A with complex entries, the field of values F is defined as

$$F(A) \equiv \{X^*AX : X \in \mathbb{C}^n \text{ and } \|X\| = 1\}, \quad (1)$$

where X^* denotes the conjugate transpose of X. $F(A)$ is a connected, convex and compact subset of $\mathbb{C}$. It can be thought of as a picture of the matrix that provides useful information about it. [3] For example, the $F(A)$ of a Hermitian matrix is a line segment on the real line, whereas for a normal matrix, in general, is a polygon. A learning rule, and a generalization, that computes eigenvectors and eigenvalues of normal matrices will be derived. Properties of the field of values and neural techniques, such as gradient ascent and constrained optimization, were used in its development. Necessarily the backdrop of the learning rule is the complex domain.

George M. Georgiou is with the School of Computer Science and Engineering, California State University, San Bernardino, CA 92407, USA; email: georgiou@csusb.edu.

Complex-valued neural networks is an active area of research with many applications. [4]–[10] While in most cases complex-valued connection matrices have been taken to be Hermitian, e.g. that of the complex Hopfield network, there has been early work in the area that suggested that more general complex connection matrices provide a richer set of dynamic behavior. [11] [12] The present work mainly deals with normal matrices, which includes Hermitian matrices as a subset, and in particular in the iterative computation of eigenvalues and eigenvectors of normal matrices.

There is a plethora algorithms in neural networks which involve computing the eigenvectors and eigenvalues of matrices. Some well known examples include Oja's rule [13], for extracting the principal eigenvector of the correlation matrix of the inputs, and Sanger's algorithm [14] and the APEX algorithm [15], [16], the last two of which extract multiple eigenvectors by employing lateral connections at the output neurons, implicitly or explicitly. A fairly inclusive and comparative discussion of such algorithms can be found in [10]. In addition, the complex-valued counterparts of the algorithms are also found in the same reference. Some examples of neural computation (in the real domain) that compute eigenpairs of given matrices include [17]–[20].

A. The field of values

We will look into the properties of the field of values of an $n \times n$ square matrix with complex entries, which we denote by $M_n(\mathbb{C})$, or simply by M_n.

Several definitions and results from the theory of matrices will be needed and will be given below. Proofs of the theorems and other information on $F(A)$ can be found in these references [1], [2].

Theorem 1: Let A be in M_n. A is normal if and only if there exist unitary matrix U and diagonal matrix D such that $A = UDU^*$.
In this previous theorem, the diagonal entries of D are the eigenvalues of A, and the columns of U are the corresponding eigenvectors.

Definition 1: Let A in M_n be a given matrix. The *field of values* or *numerical range* of A is defined to be the subset of the complex plane

$$F(A) \equiv \{X^*AX : X \in \mathbb{C}^n \text{ and } \|X\| = 1\}. \quad (2)$$

Thus the field of values is the set of points X^*AX on the complex plane when X takes all values on the unit sphere.

Of great importance is the nature and shape of $F(A)$, which are discussed next.

B. The geometry of $F(A)$

Here we present properties of the field of values that are useful in understanding the geometry of $F(A)$.

Theorem 2: Let A be in M_n. Then, $F(A)$ is a compact and convex subset of the complex plane.

We denote the spectrum of a matrix $A \in M_n$, that is, its set of eigenvalues, by $\sigma(A)$, and the convex hull of a set $S \subset \mathbb{C}$ by $Co(S)$.

Property 1: If $\alpha \in \mathbb{C}, F(\alpha A) = \alpha F(A)$.

Property 2: If $\beta \in \mathbb{C}$ and I is the identity matrix, $F(A + \beta I) = F(A) + \beta$.

Property 3 (Normality): If A is a normal matrix in M_n, then $F(A) = Co(\sigma(A))$.

Since $\sigma(A)$ is a finite set of points, at most n, the field of values $F(A)$ of normal matrix A is a polygon, possibly collapsed to a line segment or to a point. Since all eigenvalues of a Hermitian matrix are real, and Hermitian matrices are normal, it can be concluded that their field of values is a line segment on the real line, with the endpoints being the extreme eigenvalues.

It has been proven that for any $A \in M_2$, $F(A)$ is an ellipse, which, however, could degenerate to a line segment or a point. For $n \geq 3$ dimensions, there is a great variety of shapes of $F(\cdot)$. However, there no general characterization. [2, p. 48]

The *direct sum* of two matrices, $A \oplus B$, $A \in M_k$ and $B \in M_l$, is a new matrix $C \in M_{k+l}$ formed by placing the matrices so that their main diagonals taken together are the diagonal of C, and the remaining entries are zero:

$$C = A \oplus B \equiv \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \quad (3)$$

Property 4: For all $A \in M_k$ and $B \in M_l$

$$F(A \oplus B) = Co(F(A) \cup F(B)). \quad (4)$$

The latter property gives us a means of constructing complicated $F(\cdot)$ from simpler ones.

Definition 2: Let $A \in M_n$. A point α on the boundary of $F(A)$ is called a *sharp point* [2, p. 50] if there are angles θ_1 and θ_2 such with $0 \leq \theta_1 < \theta_2 \leq \pi$ for which

$$\Re(e^{i\theta}\alpha) = \max\{\Re(\beta) : \beta \in F(e^{i\theta}A) \text{ for all } \theta \in (\theta_1, \theta_2)\}, \quad (5)$$

where $\Re(\cdot)$ indicates real part.

Intuitively, a sharp point is "corner" on the boundary of the field of values; that is, a point where there are two tangent vectors, depending on the direction of approach. Thus, the polygon vertices of the field of values of a normal matrix, its eigenvalues, are sharp points. An eigenvalue that happens to be collinear with two adjacent ones will not be a sharp point. Sharp points are called *extreme* points in [1], where an alternative definition is used.

Theorem 3: If α is a sharp point of $F(A)$, where $A \in \mathbb{M}_n$, then α is an eigenvalue of A.

The next theorem characterizes sharp points.

Theorem 4: Let $A \in \mathbb{M}_n$ and α be a sharp point of $F(A)$. Then, the unit vector X for which $\alpha = X^*AX$ is an eigenvector of A. [2, p. 55]

If A is a normal matrix with distinct eigenvalues, the last two theorems imply that if we find a unit vector X such that X^*AX is a vertex of $F(A)$, then X^*AX is an eigenvalue of A and X is the corresponding eigenvector.

C. Examples of field of values

Plotting of the field of values of matrices can be done by generating its boundary point-by-point, and connecting the points. [2, p. 33] Figures 1 through 4 show the field of values of various matrices. The $+$ signs in the plots indicate the eigenvalues.

D. Constructing normal matrices

A normal matrix with a given set of eigenvalues and corresponding eigenvectors can be constructed by using the unitary decomposition of normal matrices (Theorem 1):

$$A = UDU^* \quad (6)$$

The columns of unitary matrix U are the normalized (orthogonal) eigenvectors and D is a diagonal matrix with entries being the corresponding eigenvalues.

To construct a normal matrix with given eigenvalues and random eigenvectors, the QR factorization [21] can be used. Given a matrix N with random entries, it can be decomposed as $N = QR$, where Q is a unitary matrix and R is an upper triangular matrix. Matrix R is discarded. Using Q in the place of U in Equation (6), normal matrix A is obtained.

II. The new learning rule

A. Description of the proposed method

A new learning algorithm is derived which can be used to find the eigenvectors of a given normal matrix. The

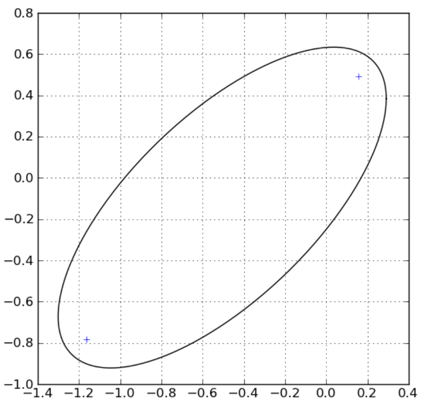

Fig. 1. $F(A), A \in \mathbb{M}_2$. A is random.

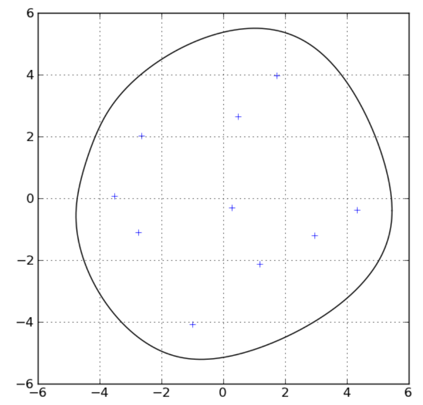

Fig. 2. $F(A), A \in \mathbb{M}_{10}$. A is random.

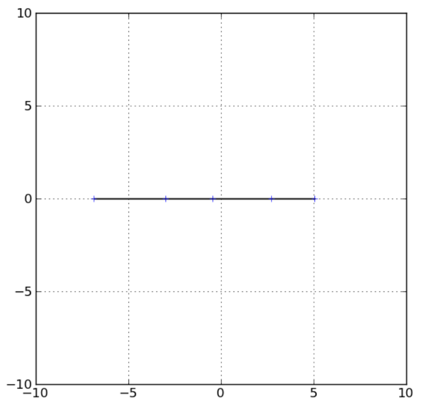

Fig. 3. $F(A), A \in \mathbb{M}_5$. A is Hermitian.

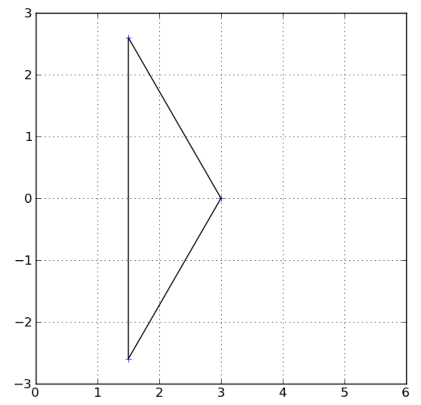

Fig. 4. $F(A), A \in \mathbb{M}_3$. A is normal.

algorithm works best for normal matrices that have distinct eigenvalues and no three of their eigenvalues lie on a line on the complex plane. The algorithm convergences to corners in the field of values. For example, Figures 1 and 2 show that all eigenvalues that are internal to the field of values. The learning rule presented here will not converge to any of the eigenvalues of the corresponding (non-normal) matrices.

Theorem 5: Let f be a convex function defined on the bounded, closed convex set Ω. If f has a maximum over Ω, it is achieved at an extreme point of Ω. [22, p. 198]

Since the field of values $F(C)$ is a (polygonal) bounded, compact convex set, and $f(z) = z\,\overline{z} = |z|^2$, $z \in F(C)$, is a convex function, $f(z)$ is maximized when z is an extreme point (a corner) of $F(C)$. The overbar denotes complex conjugation. Note that $f(z)$ is simply the square of the Euclidean distance of point $z \equiv Y^*CY$ from the origin, for some $Y \in \mathbb{C}^n, \|Y\| = 1$.

We use constrained gradient ascent to obtain Y that solves this optimization problem:

$$\max q\,\overline{q}, \qquad \|Y\| = 1, \tag{7}$$

where we define $q = Y^*CY$. Using λ as a Lagrange multiplier, the problem now becomes maximizing function g:

$$g(Y) = q\,\overline{q} - \lambda(Y^*Y - 1). \tag{8}$$

The (complex) gradient of function $g : \mathbb{C}^n \to \mathbb{C}^n$ with respect to its argument $Y \in \mathbb{C}^n$ is defined as

$$\nabla_Y g(Y) \equiv \nabla_{Y_R} g(Y_R) + i\,\nabla_{Y_I} g(Y_I), \tag{9}$$

where Y is split into real and imaginary parts: $Y = Y_R + i\,Y_I$, with $Y_R, Y_I \in \mathbb{R}^n$. The gradients in the RHS of (9), involving vectors in $\mathbb{R}^n$, have the usual meaning. As usual, $i = \sqrt{-1}$.

The gradient $\nabla_Y g(Y)$, which will provide the basis for the learning rule, is derived to be

$$\nabla_Y g(Y) = 2(qC^* + \overline{q}C)Y - 2\lambda Y. \tag{10}$$

Proof: The first term in the RHS will first be derived. We define $E = q\,\overline{q}$

$$\nabla_Y E = 2(qC^* + \overline{q}C)Y. \tag{11}$$

Writing Y in terms of its real and imaginary parts, E is expanded as follows:

$$\begin{aligned} E &= q\,\overline{q} \\ &= (Y^*CY)\overline{(Y^*CY)} \\ &= (Y^*CY)(Y^T\overline{CY}) \\ &= (Y_R^T - iY_I^T)C(Y_R^T + iY_I^T) \\ &\quad (Y_R^T + iY_I^T)\overline{C}(Y_R^T - iY_I^T) \end{aligned} \tag{12}$$

The gradient of Equation (11) becomes:

$$\begin{aligned} \nabla_Y(E) &= \frac{\partial E}{\partial Y_R} + i\,\frac{\partial E}{\partial Y_I} && (13)\\ &= (q\frac{\partial \overline{q}}{\partial Y_R} + \frac{\partial q}{\partial Y_R}\overline{q}) + i\,(q\frac{\partial \overline{q}}{\partial Y_I} + \frac{\partial q}{\partial Y_I}\overline{q}) && (14)\\ &= q((\overline{C} + \overline{C}^T)Y_R + i\,\overline{C}^TY_I - i\,\overline{C}Y_I) + \\ &\quad \overline{q}((C + \overline{C})Y_R + i\,CY_I - i\,C^TY_I)) + \\ &\quad i\,(q(i\,\overline{C}Y_R - i\,\overline{C}^TY_R + (\overline{C} + \overline{C}^T)X_I) + \\ &\quad \overline{q}(i\,C^TY_R - i\,CY_R + (C + C^T))) && (15)\\ &= q(2C^*Y) + \overline{q}(2CY) && (16)\\ &= 2(qC^* + \overline{q}C)Y. && (17) \end{aligned}$$

The second term of the RHS in Equation (8) is derived by expanding Y:

$$Y = (Y_{1R} + i\,Y_{1I}, Y_{2R} + i\,Y_{2I}, \ldots, Y_{nR} + i\,Y_{nI})^T. \tag{18}$$

$$\begin{aligned} \nabla_Y(Y^*Y) &= \nabla_Y(Y_{1R}^2 + Y_{1I}^2 + Y_{2R}^2 + Y_{2I}^2 + \ldots + \\ &\quad Y_{nR}^2 + Y_{nI}^2) && (19)\\ &= 2Y && (20) \end{aligned}$$

■

At equilibrium, the gradient in Equation (10) equals zero, and $Y^*Y = 1$. Setting Equation (10) to zero and left multiplying it by Y^*, we have

$$2qY^*C^*Y + 2\overline{q}Y^*CY = 2\lambda Y^*Y. \tag{21}$$

Using the fact that that $Y^*C^*Y = (Y^*CY)^* = \overline{q}$, the previous equation becomes

$$\lambda = 2q\overline{q}. \tag{22}$$

Substituting λ in Equation (10), we can arrive at this gradient learning rule:

$$\Delta Y = \alpha\,((qC^* + \overline{q}C)Y - 2q\overline{q}Y), \tag{23}$$

where α is a small positive constant, the learning rate. A factor of 2 has been absorbed in it. An alternative derivation of (23), using Wirtinger calculus, appears in the Appendix. The first term in (23) is for maximization, while the second is for normalization. In neural networks, the classic example of maximization with the normalization constraint is Oja's rule [13]. Its complex version can be found in this reference [10]. While Oja's rule also consists of a maximization and a normalization factor, it differs substantially in how it is applied from Equation (23): in Oja's rule the input is varied and the weights converge to the principal component, i.e. the eigenvector that corresponds to the largest eigenvalue of the correlation matrix of the inputs. In the derived rule above once the initial value of Y is given, the algorithm modifies Y until convergence. If C is a normal matrix, in general, it will converge to an eigenvector Y, and the corresponding eigenvalue will be $q = Y^*CY$.

By rewriting Equation (23) as

$$\begin{aligned} \Delta Y &= \alpha\,((qC^* + \overline{q}C) - 2q\overline{q}I)Y \\ &= \alpha\,(q(C^* - \overline{q}I) + \overline{q}(C - qI))Y, \end{aligned} \tag{24}$$

where $I \in \mathbb{M}_n$ is the identity matrix, we note that the quantity by which Y is multiplied is a Hermitian matrix. This follows from the fact that q is a complex scalar and that for any matrix $A \in M_n$, $A + A^*$ is Hermitian.

The algorithm of Equation (23), being a gradient optimization method, depends on the initial value of Y. It is possible that, as is the case of the power method algorithm [23], [24], if Y is initialized so that it does not have a component in the direction of an eigenvector, it will not converge to an eigenpair. To understand this scenario, consider a real C and a real initial vector Y: no imaginary component be generated for Y and q in the course of the algorithm, even if all eigenpairs may have imaginary components. However, numerical roundoff errors may contribute to such a missing component, and the algorithm may work after all. It is best if the real and imaginary parts of the components of Y are initialized with non-zero values .

The algorithm will tend to converge to eigenvalues that are far away from zero. This is explained by the fact that the learning rule tries to maximize the distance of q from the origin.

A more general learning rule can be derived that will maximize the distance of q from an arbitrary point $c \in \mathbb{C}$. Equation (8) becomes

$$\hat{g}(Y) = (q - c)\,\overline{(q - c)} - \lambda(Y^*Y - 1). \tag{25}$$

Following similar steps as in the derivation of the previous learning rule, the new learning rule can be derived as:

$$\begin{aligned}\Delta Y &= \alpha\,(((q-c)C^* + (\overline{q}-\overline{c})C)Y - \\ &\quad (q(\overline{q}-\overline{c}) + \overline{q}(q-c))Y) \\ &= \alpha\,(((q-c)C^* + (\overline{q}-\overline{c})C) - \\ &\quad (q(\overline{q}-\overline{c}) + \overline{q}(q-c))I)Y) \\ &= \alpha\,((q-c)(C^* - \overline{q}I) + \\ &\quad (\overline{q}-\overline{c})(C-qI))Y \end{aligned} \tag{26}$$

Obviously, when c is at the origin, this learning rule collapses to the earlier one. The quantity by which Y is multiplied with is, again, a Hermitian matrix. When the goal is to find as many eigenvalues and eigenvectors of the normal matrix C as possible, it is best to pick c such that it is in the interior of the field of values. Otherwise, as above, the algorithm may never converge to eigenvalues that are close to c: simply the algorithm favors those that are far away, unless, of course, it gets stuck in a local minimum, manifested as an eigenvalue closer in distance from c. A natural choice for c is the average value of the trace of matrix C:

$$c = \frac{\sum_{i=1}^{n} \lambda_i}{n} = \frac{\mathrm{Tr}(C)}{n}, \tag{27}$$

where λ_i's are the eigenvalues, including multiplicities, and $C \in \mathbb{M}_n$. If C is normal and the eigenvalues are distinct, i.e. each will be a vertex of a polygon that contains the field of values $F(C)$, c will be the centroid of $F(C)$.

An alternative method to avoid using the learning rule of Equation (26) is to use a new matrix $\hat{C} = C - cI$. This transformation is being used in the shifted power iteration and shifted inverse iteration methods of computing eigenvalues and eigenvectors. [24] Then, by Property 2, $F(\hat{C}) = F(C) - c$. Point c in $F(C)$ will be translated to the origin. The original learning rule of Equation (23) can be used. Note if the eigenvalues of C are $\lambda_0, \lambda_1, \ldots, \lambda_n$, the eigenvalues of $\hat{C}$ are $\lambda_0 - c, \lambda_1 - c, \ldots, \lambda_n - c$. The respective eigenvectors of C and $\hat{C}$ are the same.

Proof: Let λ_i be an eigenvalue of C, and the corresponding eigenvector be Y_i. Then, $CY_i = \lambda_i Y_i$.

$$\hat{C}Y_i = (C - cI)Y_i = CY_i - cY_i = \lambda_i Y_i - cY_i = (\lambda_i - c)Y_i \tag{28}$$

Hence, $\lambda_i - c$ is an eigenvalue of $\hat{C}$ and Y_i is the corresponding eigenvector. ■

This translation (shifting) method can be viewed and used as a deflation process: once an eigenpair (λ_i, Y_i) of C is found, the new matrix $\hat{C} = C - \lambda_i I$ is formed. Since λ_i has been translated to the origin, the learning rule of Equation (10) will not converge to it again, and another eigenpair will be computed. However, this translation cannot be used simultaneously with multiple distinct eigenvalues since only one eigenvalue can be translated at the origin at one time.

B. *Discussion of new method in relation to other methods of finding eigenpairs*

The new method (Equation (23)) starts from a random initial vector and iterates until it converges to an eigenvector; the corresponding eigenvalue can then be computed. At each iteration, the method multiplies a matrix with the Y vector and applies a normalizing term. It resembles the power method, which finds the dominant eigenpair. At every iteration, the power method computes vector $Y \leftarrow CY$ and then explicitly normalizes Y. Eventually, it converges to the eigenvector that corresponds to the dominant eigenvalue, which is required to be a single one for the method to work. As shown in the results section, the proposed method converges to eigenpairs even when all eigenvalues have the same magnitude, e.g. the case in Figure 6, depending on the initial value of Y and the geometry of the field of values. Hence, the aims of the two methods are different.

Another method that starts from an initial vector and iterates until convergence is Newton's method for finding eigenpairs. [25], [26] Newton's method requires that at each iteration an $(n+1) \times (n+1)$ system of linear equations is solved, or equivalently, to invert a matrix of the same dimensions. This complexity makes the method less neural-like, and it is not the aim of the paper to compare it with the new method.

When the goal is to find all eigenpairs of matrices conventional techniques, such as the QR or the Jacobi method [23] are better suited for the task.

III. Eigenvalues and Eigenvectors as Stored Memories

Since both learning rules, Equations (23) and (28), converges to both eigenvectors and eigenvalues, these two quantities can serve as the stored memories. We note that the eigenvectors that correspond to distinct eigenvalues of normal matrices are orthogonal. It is well-known that correlation matrix memory and the discrete Hopfield neural network can recall, in general, orthogonal vector memories perfectly. [27], [28] However the basin of attraction and the trajectory of attraction cannot be easily visualized. The present method, using the field of values, provides a convenient way to visualize the dynamics of the learning rule in 2-D. The trajectory of the value $q = Y^*CY$, given an initial value of

Y, on the complex plane, reveals both qualitative and quantitative dynamic information of the process. This type of information is limited to 1-D in systems where the matrix is Hermitian (or symmetric in the real case), which is the common case in Hopfield networks and matrix associative memories. Dynamics on 1-d is not conducive to visualization. The complex plane provides an appropriate medium for visualization.

Let $X = (X_1, X_2, \dots, X_n)$ in M_n, where the X_i's are the (column) vectors to be associated (stored) in the normal matrix $C \in \mathbb{M}_n$. It is assumed that the X_i's are linearly independent. We associate each X_i with column vector U_i through this linear mapping:

$$X = PU \tag{29}$$

where $U = (U_1, U_2, \dots, U_n) \in M_n$ is unitary, and $P \in M_n$. Unitary matrix U can be chosen in a variety of ways: it may be random, the unitary matrix in the polar decomposition of X, a Hadamard matrix (a unitary matrix with entries in $\{1, -1\}$), etc.

The next step is to construct a normal matrix $C \in M_n$ having the U_i's as its eigenvectors. For that purpose we use the unitary decomposition of normal matrices:

$$C = UDU^*, \tag{30}$$

where $D \in M_n$ is a diagonal matrix of which the $d_i \equiv d_{i,i}$ entry is an eigenvalue of C with corresponding eigenvector U_i. Hence, X_i is associated with d_i as well.

The eigenvalues d_i of C are the attractors during the recall process. If it is desirable that the memories are stored in a symmetric way, which avoids undue bias toward any memory during recall. For this purpose, we choose d_i to be the n roots of unity to be the eigenvalues:

$$d_i = e^{i\frac{2\pi k}{n}}, k = 0, 1, \dots, n-1. \tag{31}$$

The stored information is in the set $\{P^{-1}, C\}$. Matrix P is invertible due to the linear independence of the X_i.

The recall process is best observed in the field of values $F(C)$.

Suppose that $X_l \in \mathbb{C}^n$ is a given probe vector, and it is desired to find the closest stored vector X_k.

We first form $Y_l \in \mathbb{C}^n$

$$Y_l = P^{-1}X_l. \tag{32}$$

Vector Y_l is used as the initialization vector in either one of the learning rules, (23) or (26). Once the process converges to an eigenvector Y_i of C, the recalled vector is $X_i = PY_i$.

IV. Results

To explore the efficacy of the learning rule of Equation (23) several experiments were conducted. In general the algorithm was shown to be quite robust. As long as the learning rate α was sufficiently small, the algorithm converged to an eigenvalue and the corresponding eigenvector. Beginning with Figure 5, each eigenvalue (vertex) is color coded. The initializing vector corresponds the point furthest away from the eigenvalue (vertex of the polygon), and as the algorithm progresses, the initial point follows a trajectory that ends up at the eigenvalue. This color scheme is akin to the one used to described the fractal behavior of Newton's method in finding roots of polynomials. [29] Each root has a different color which is the same for points in its region of attraction.

Figure 5 shows the trajectories of 500 runs of the learning rule in Equation (23). The distance of the final q to the actual eigenvalue, the error, was found to be less than 10^{-5} on the average. For each run, normal matrix $C \in M_6$ was generated with random eigenvectors and given eigenvalues (Section I-D). The vertices of the regular hexagon are the eigenvalues of C, which are $e^{i\frac{2\pi k}{6}}, k = 0, 1, \dots, 5$; i.e. six 6-th roots of unity. The hexagon itself and its interior make up the field of values of C. Each initial Y was randomly generated and its trajectory has the color of the eigenvalue it converged to. The value of the learning rate used was $\alpha = 0.01$. Each run was stopped after 500 iterations.

To clearly show the regions of attraction of each eigenvalue, the algorithm was run 5,000 times, each again with a random initial Y, while all other variables remained the same. The result is shown in Figure 6. The regions of attraction are well defined, however, as it can be seen especially from Figure 6, they are not exclusive. On occasion, an initial value of q happens to follow a trajectory away from the closest eigenvalue. Figures 7 ($C \in \mathbb{M}_6$) and 8 ($C \in \mathbb{M}_5$) each show 100 runs of the same rule (Equation (23)). The initial Ys were randomly generated. For each run, matrix C was generated with random eigenvectors and as eigenvalues the 6-th and 5-th roots of unity, for the respective figures. From the trajectories, it can be seen that eigenvalues (and their corresponding eigenvectors) that are furthest away from the origin are favored by the algorithm to converge to. This can be explained from the fact the learning rule tries to maximize the distance of q from the origin. Eigenvalues that are not at a maximal distance from the origin and yet the algorithm converges to them, correspond to local minima. Figures 9 and 10 show 100 runs of the generalized learning rule of Equation (26). The values of c are $0.5 + i\ 0.5$ and $-1 - i$, respectively. While

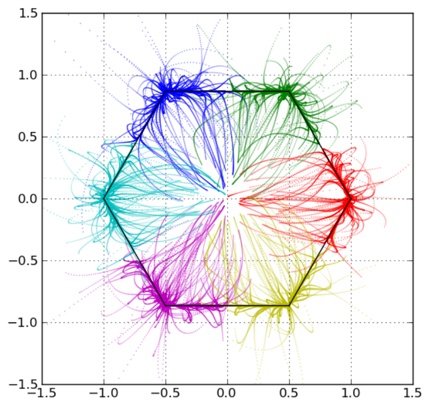
Fig. 5. The trajectories of 500 runs of the algorithm.

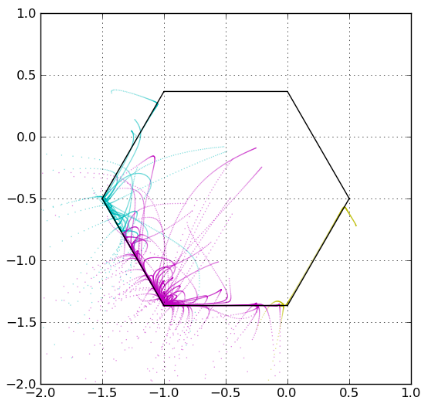
Fig. 7. The trajectories of 100 runs of the algorithm.

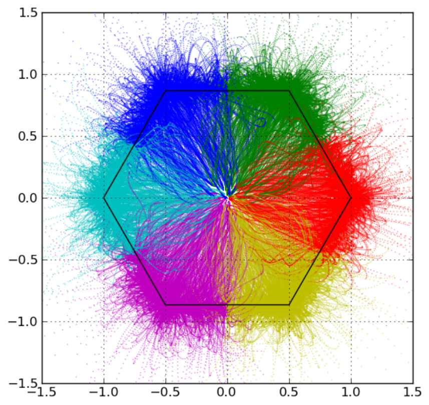
Fig. 6. The trajectories of 5,000 runs of the algorithm.

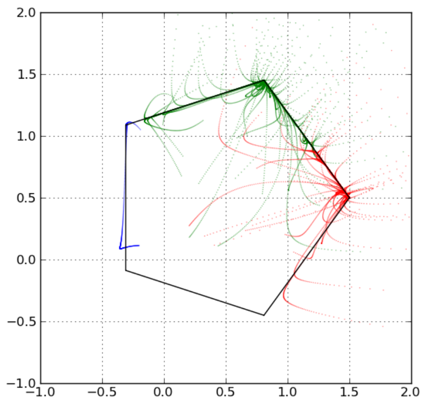
Fig. 8. The trajectories of 100 runs of the algorithm.

the trajectories and convergence behavior in general are comparable to first rule, an anomaly was observed: when the origin was outside the field of values, the algorithm sometimes converged to it, even though it was not an eigenvalue. This can be seen in Figure 10. The final Y values in those cases were the zero vector or, rarely, very close to it.

For higher dimensions, polygons approach smooth curves, and it becomes difficult to distinguish the individual eigenvalues (vertices). For example, Figure 11 shows a regular polygon with 100 vertices which approaches a circle. The vertices and the corresponding regions of attraction are not easily distinguishable. Each run (point in the trajectory) was stopped after 1,000 iterations.

The method appears to be quite robust in the sense that as long as the learning rate was small (0.01 in most our cases) and the values of matrix entries and the initialization weights were small, each component less than 1, the algorithm was well-behaved, i.e. it converged without becoming unstable.

It is noted that if at any time in the course of the two algorithms Y takes the value of the zero vector, Y cannot change; it is stuck at the zero vector. It seems unlikely for this to happen with the learning rule of Equation (23), since q is specifically designed to move away from the origin. This is not the case in the generalized algorithm, which is designed for q to move away from point c. While more analysis and experiments are needed to further explain the anomalous behavior of convergence to zero, it seems clear that the first algorithm is preferable. Instead of using the generalized algorithm, one can use the translation method together with the first algorithm, as explained at the end of Section II.

V. Conclusion

Although the field of values of a matrix is well-known in mathematics, it does not appear to have been used in any significant way in neural networks or in other fields of engineering. Using neural networks techniques and the field of values, the two algorithms which were introduced find eigenpairs of normal matrices. It has been

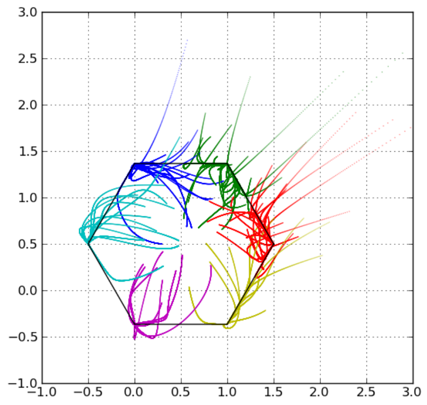

Fig. 9. The trajectories of 100 runs of the generalized algorithm, with $c = 0.5 + i\,0.5$.

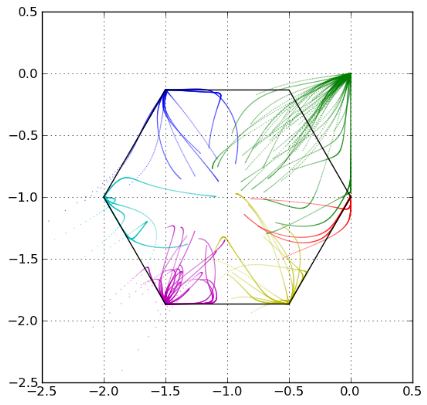

Fig. 10. The trajectories of 100 runs of the generalized algorithm, with $c = -1 - i$.

demonstrated that the generalized algorithm sometimes converges to the origin without it being an eigenvalue. This aberrant behavior was exhibited when the origin was outside the field of values. The first algorithm, which maximizes the distance of q from the origin, experimentally was shown to be stable, converging to an eigenpair given that the initial vector was random and the learning sufficiently small.

Appendix
Alternative Derivation of Learning Rule using Wirtinger's Calculus

We derive the learning rule of Equation (23) using Wirtinger calculus. [30], [31] We would like to maximize function g in Equation (8):

$$g(Y, Y^*) \;=\; q\,\overline{q} - \lambda(Y^*Y - 1). \tag{33}$$

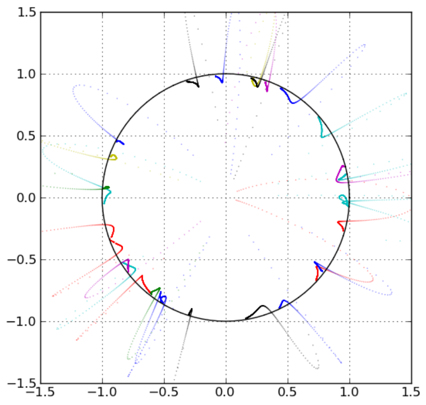

Fig. 11. The trajectories of 30 runs on a regular polygon with 100 vertices.

The Wirtinger gradient $\nabla_{Y^*}(\cdot) \equiv \nabla$ is as follows:

$$\begin{aligned}\nabla g &= q\,\nabla\overline{q} + \overline{q}\,\nabla q - \lambda Y \\ &= q\,\nabla\overline{Y^*CY} + \overline{q}\,CY - \lambda Y \\ &= q\,\nabla(Y^*C^*Y) + \overline{q}\,CY - \lambda Y \\ &= q\,C^*Y + \overline{q}\,CY - \lambda Y. \end{aligned} \tag{34}$$

At equilibrium the above equation will be equal to zero and $Y^*Y = 1$. Setting the equation to zero and left multiplying by Y^*, we proceed as in Equations (21) and (22) (without the factor 2, which it is absorbed in the Wirtinger gradient):

$$qY^*C^*Y + \overline{q}Y^*CY = \lambda Y^*Y. \tag{35}$$

$$\lambda = 2q\overline{q}. \tag{36}$$

Finally, we arrive at the learning rule of Equation (23) by considering the gradient in (34):

$$\Delta Y = \alpha\left((qC^* + \overline{q}C)Y - 2q\overline{q}Y\right). \tag{37}$$

References

[1] K. E. Gustafson and D. K. Rao, *Numerical Range: The Field of Values of Linear Operators and Matrices (Universitext)*, 1st ed. Springer, 11 1996.

[2] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*. Cambridge University Press, 5 1991.

[3] L. Hogben, Ed., *Handbook of Linear Algebra*, 1st ed. Chapman and Hall/CRC, 11 2006.

[4] A. Hirose, Ed., *Complex-Valued Neural Networks: Theories and Applications*. World Scientific Publishing, 2003.

[5] A. Hirose, *Complex-Valued Neural Networks*, ser. Studies in Computational Intelligence. Springer, 2006, vol. 32.

[6] ——, *Complex-valued Neural networks*. Saiensu-sha, 2005, in Japanese.

[7] T. Nitta, *Complex-valued Neural Networks: Utilizing High-dimensional Parameters*, 1st ed. Information Science Reference, 1 2009.

[8] I. N. Aizenberg, *Complex-Valued Neural Networks with Multi-Valued Neurons*, ser. Studies in Computational Intelligence. Springer, 2011, vol. 353.

[9] S. Suresh, N. Sundararajan, and R. Savitha, *Supervised Learning with Complex-valued Neural Networks*, ser. Studies in Computational Intelligence. Springer, 2013, vol. 421.

[10] Z. Chen, S. Haykin, J. J. Eggermont, and S. Becker, *Correlative Learning: A Basis for Brain and Adaptive Systems (Adaptive and Learning Systems for Signal Processing, Communications and Control Series)*, 1st ed. Wiley-Interscience, 10 2007.

[11] A. Hirose, "Fractal variation of attractors in complex-valued neural networks," *Neural Processing Letters*, vol. 1, no. 1, pp. 6–8, 1994.

[12] ——, "Non-hermitian associative memories spontaneously generating dynamical attractors," in *Neural Networks, 1993. IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on*, vol. 3, oct. 1993, pp. 2567 – 2570 vol.3.

[13] E. Oja, "A simplified neuron model as a principal component analyzer," *J. Math. Biology*, vol. 15, pp. 267–273, 1982.

[14] T. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Networks*, vol. 2, pp. 459–473, 1989.

[15] K. I. Diamantaras and S.-Y. Kung, *Principal Component Neural Networks: Theory and Applications*, ser. Wiley Series on Adaptive and Learning Systems for Signal Processing, Communications, and Control. New York: Wiley, 1996.

[16] S.-Y. Kung, K. I. Diamantaras, and J.-S. Taur, "Adaptive principal component EXtraction (APEX) and applications," *IEEE Transactions on Signal Processing*, vol. 42, no. 5, pp. 1202–1217, 1994.

[17] Y. Liu, Z. You, and L. Cao, "A concise functional neural network computing the largest (smallest) eigenvalue and one corresponding eigenvector of a real symmetric matrix," in *Neural Networks and Brain, 2005. ICNN B '05. International Conference on*, vol. 3, oct. 2005, pp. 1334 –1339.

[18] Q. Zhang, F. Feng, and F. Liu, "Neurodynamic approach for generalized eigenvalue problems," in *Computational Intelligence and Security, 2006 International Conference on*, vol. 1, nov. 2006, pp. 345 –350.

[19] Y. Shuijun and L. Diannong, "Neural network based approach for eigenstructure extraction," in *Aerospace and Electronics Conference, 1995. NAECON 1995., Proceedings of the IEEE 1995 National*, vol. 1, may 1995, pp. 102 –105 vol.1.

[20] T. Ying and H. Zhenya, "Neural network approaches for the extraction of the eigenstructure," in *Neural Networks for Signal Processing [1996] VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop*, sep 1996, pp. 23 –32.

[21] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, 12 1985.

[22] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*, 3rd ed. Springer, 2008.

[23] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[24] Y. Saad, "Numerical Methods for Large Eigenvalue Problems," *Manchester University Press*, no. Second Edition, 2011.

[25] P. Anselane and L. Rall, "The solution of characteristic value - vector problems by Newtons method," *Numerische Mathematik*, vol. 11, pp. 38–45, 1968.

[26] K. Datta, Y. Hong, and R. B. Lee, "Parametrized Newtons iteration for computing an eigenpair of a real symmetric matrix in an interval," *Comput. Methods Appl. Math.*, vol. 3, no. 4, pp. 517–535, 2003.

[27] S. O. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Prentice Hall, 11 2008.

[28] J. M. Zurada, *Introduction to Artificial Neural Systems*. St. Paul, MN: West Publishing Company, 1992.

[29] R. T. Stevens, *Fractal programming in C*. IDG Books Worldwide, Inc., 1991.

[30] D. Brandwood, "A complex gradient operator and its application in adaptive array theory," in *IEE Proceedings H (Microwaves, Optics and Antennas)*, vol. 130, no. 1. IET, 1983, pp. 11–16.

[31] M. F. Amin and K. Murase, "Learning algorithms in complex-valued neural networks using Wirtinger calculus," in *Complex-Valued Neural Networks*, A. Hirose, Ed. Wiley, 2013, pp. 75–102.

George M. Georgiou received the B.S in Electrical Engineering degree from Louisiana Tech University in 1985, the M.S. in Electrical Engineering and M.S. in Mathematics degrees from Louisiana State University in 1987 and 1988, respectively, and the M.S. and Ph.D. degrees in Computer Science from Tulane University, New Orleans, Louisiana, USA, in 1990 and 1992, respectively. Currently, he is Professor of Computer Science and Engineering, and Associate Dean of the College of Natural Sciences at California State University, San Bernardino, USA. He served as Chair and Director of the School of Computer Science and Engineering at the same institution. His Ph.D. dissertation was the first on the subject of complex-valued neural networks. His research interests include complex-valued neural networks, machine learning, linear algebra methods, and self-organizing systems.