Efficient Learning of Transform-Domain LMS Filter Using Graph Laplacian

Tamal Batabyal, Daniel Weller, *Senior Member, IEEE*, Jaideep Kapur, and Scott T. Acton, *Fellow, IEEE*

***Abstract*— Transform-domain least mean squares (TDLMS) adaptive filters encompass the class of learning algorithms where the input data are subjected to a data-independent unitary transform followed by a power normalization stage as preprocessing steps. Because conventional transformations are not data-dependent, this preconditioning procedure was shown theoretically to improve the convergence of the least mean squares (LMS) filter only for certain classes of input data. So, one can tailor the transformation to the class of data. However, in reality, if the class of input data is not known beforehand, it is difficult to decide which transformation to use. Thus, there is a need to devise a learning framework to obtain such a preconditioning transformation using input data prior to applying on the input data. It is hypothesized that the underlying topology of the data affects the selection of the transformation. With the input modeled as a weighted finite graph, our method, called preconditioning using graph (PrecoG), adaptively learns the desired transform by recursive estimation of the graph Laplacian matrix. We show the efficacy of the transform as a generalized split preconditioner on a linear system of equations and in Hebbian-LMS learning models. In terms of the improvement of the condition number after applying the transformation, PrecoG performs significantly better than the existing state-of-the-art techniques that involve unitary and nonunitary transforms.**

***Index Terms*— Graph Laplacian, graph learning, Hebb-least mean squares (LMS) learning, LMS filter, split preconditioner, unitary transform.**

I. Introduction

In 1960, Widrow and Hoff [1] proposed a class of *least mean squares* (LMS) algorithms to recursively compute the coefficients of an N-tap finite impulse response (FIR) filter that minimizes the output error signal. This computation may be achieved by a stochastic gradient descent approach where the filter coefficients are evaluated as a function of the *current error* at the output. The LMS algorithm and its variants were subsequently used in a myriad of applications, including echo cancellation [2]–[5], inverse modeling [6], system identification, signal filtering [7], [8], and several others.

Manuscript received July 23, 2021; revised December 23, 2021; accepted January 12, 2022. *(Corresponding author: Tamal Batabyal.)*

Tamal Batabyal and Jaideep Kapur are with the Department of Neurology, University of Virginia, Charlottesville, VA 22904 USA (e-mail: tb2ea@virginia.edu).

Daniel Weller was with the Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904 USA. He is now with KLA Corporation, Ann Arbor, MI 48105 USA.

Scott T. Acton is with the Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904 USA

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3144637.

Digital Object Identifier 10.1109/TNNLS.2022.3144637

Two of the major issues with the aforementioned approach are the convergence speed and stability. The filter coefficients (or weights) converge in mean while showing minor fluctuation in magnitude around the optimal values. The convergence speed depends on the condition number of the autocorrelation matrix of the input, where a condition number close to unity connotes a fast and stable convergence. Later, adaptive techniques, such as least square lattice (LSL) and gradient adaptive lattice (GAL) [9], [10] filters were designed to achieve faster convergence, immunity to a poor condition number of the input autocorrelation matrix, and better finite precision implementation compared to the LMS filter. However, these stochastic gradient filters may sometimes produce significant numerical errors, and the convergence is poor compared to recursive least squares (RLS) filters [11]. Although the RLS algorithm has increased computational complexity compared to that of the LMS algorithm, effective algorithms, such as dichotomous coordinate descent [12] for low-complexity RLS have been proposed to reduce the number of multiplication operations.

A class of affine projection algorithms and their improvements for real-time applications was proposed for faster convergence of LMS tap weights [4], [13]–[15]. In many applications, the performance of affine projection algorithms depends on the projection order, initial configuration, signal stationarity, and other factors. In general, the algorithms are attempted to substitute the expensive matrix-inverse operation. These algorithms can be easily incorporated into our proposed framework, called preconditioning using graph (PrecoG). PrecoG transforms the input before the filter operation. The question of how decorrelation of the input data using PrecoG will affect the performance of such algorithms with regard to the convergence of the LMS filter tap weights is outside the scope of this work.

Due to the fact that the nature of the autocorrelation matrix is data-dependent, improving its condition number using a transformation is a way to circumvent the convergence issue in the case of realtime data. To obtain well-conditioned autocorrelation matrix of any real-world input data, we transform the input data *a priori*. One such example of this approach is popularly known as transform-domain LMS (TDLMS) (see Section II). A schematic presentation of LMS and TDLMS filters is provided in Fig. 1(C). The discrete Fourier transform (DFT), discrete cosine transform (DCT) [16]–[19], and others act as suitable off-the-shelf transformations of the input data for such problems. The aforementioned step is immediately

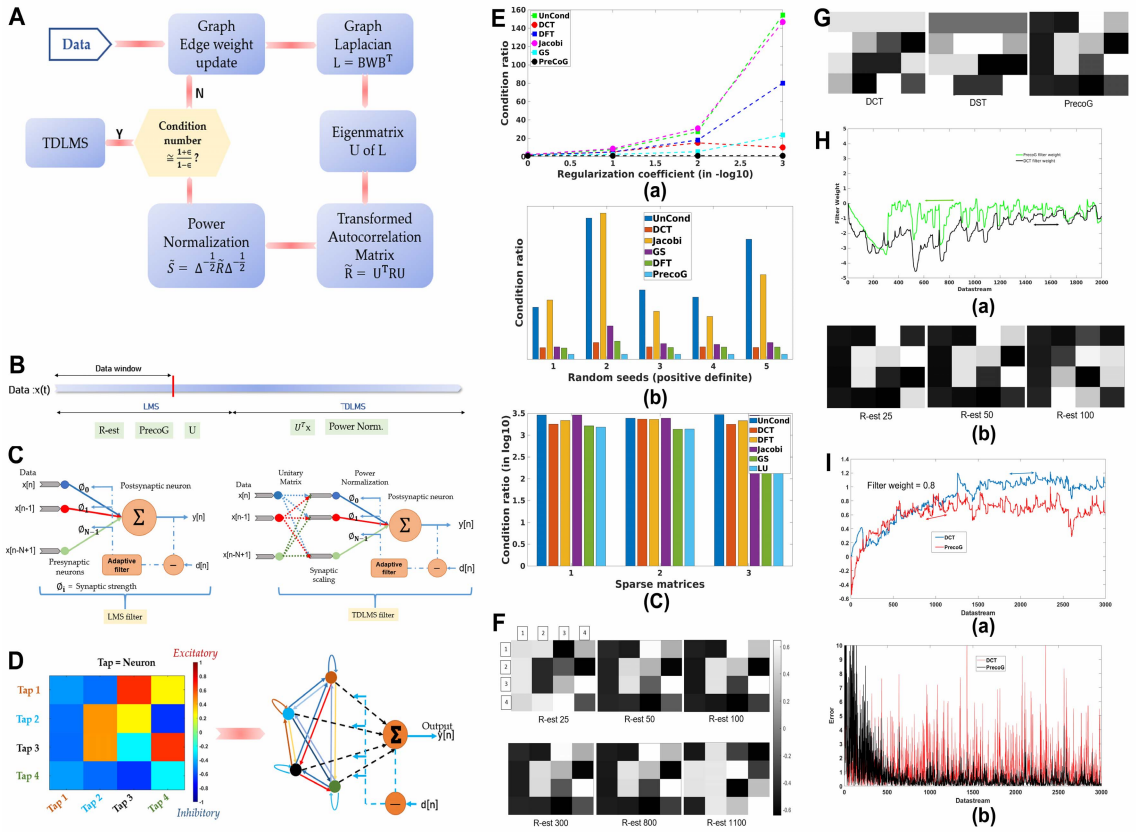

Fig. 1. (A) Schematic of our algorithm explaining how the PrecoG transformation is computed. (B) Online mode of LMS in case of PrecoG. Here, the autocorrelation is estimated using an initial window of data. During that time, classical LMS is executed. Once the U_N is computed, subsequent data are processed in TDLMS mode. The input graph is also show here. (C) Equivalence between biological neurons and its simplified computational analog in LMS. Synaptic scaling is achieved via power normalization stage. (D) Preconditioning matrix U is shown. Color indicates the strength of excitation or, equivalently, inhibition. The interaction (not the input graph) among four neurons is also shown in the graph. (E) Condition ratios obtained by applying the algorithms on (a) regularized Hilbert matrices with varied regularization parameters, (b) a set of random matrices containing entries $\sim Gaussian(0, 1)$, and (c) random matrices of varied sparsity (for sparse linear systems). (F) PrecoGs (U_N) which are estimated at different lengths of initial data window. All matrices are orthonormal. Given different windows of data, the estimated autocorrelations are different, leading to different U_Ns. The excitatory (more white) and inhibitory (more black) behavior that drives by each filter tap (a neuron) onto its neighboring taps are evident. It is evident that PrecoG has a considerable magnitude of inhibitory strength among four neurons compared to conventional transforms. However, after passing a considerable amount of data (for example, PrecoG using 1100 1-D data samples tagged as R-est 1100 in the figure) to estimate PrecoG, the inhibition breaks down, leading to more excitatory strength. A list of condition numbers when these matrices (U_N) are applied to the input is given in Table I. (G) Figure shows the preconditioners—DCT, DST, and PrecoG (125 window length, AR(1) with $\rho = 0.9$). (H) (a) Plot showing the convergence of a tap weight in cases of DCT (black) and PrecoG (green), keeping the values of all the parameters same for both cases. The double-headed arrows mark the iteration where the weights started converging to the actual filter tap weight in each case. It shows that the LMS tap weight converges at 640th iteration in case of PrecoG when compared to 1590th iteration in case of DCT. (b) PrecoG transformation matrices in case of an AR(1) process with $\rho = 0.2$, estimated at different window length are shown. That because the data are comparatively decorrelated, PrecoG transformation contains pronounced inhibitory drives with small window length. AR(1) with higher correlated datastream demands longer window length. (I) (a) Plot showing the convergence of a filter weight in cases of the DCT and PrecoG when applied on an AR(2) process with $\rho_1 = 0.6$ and $\rho_2 = 0.9$. The double-headed arrows indicate the iteration interval where the tap weight starts converging. (b) Plot showing the error as the datastream progresses. With PrecoG, the error is rapidly diminished. With the DCT, the error is reduced at first and then sustains. The values of the parameters in both the experiments were kept the same.

followed by a power normalization stage [20], [21] and then used as input to the LMS filter. As a geometrical interpretation, the unitary transformation rotates the mean square error (mse) hyperellipsoid without changing its shape on the axes of LMS filter weights [21]. The rotation attempts to align the axes of the hyperellipsoid to the axes of weights. The power normalization is crucial in enhancing the speed of convergence for the LMS filter. The normalization forces the hyperellipsoid to cross all the axes at equal distance from the center of the hyperellipsoid. For a perfect alignment after the transformation, the normalization step turns the mse hyperellipsoid into a hypersphere [21]. It is important to note that these off-the-shelf transformations are still not data-dependent. They work relatively well for certain classes of data having autocorrelation matrices with special structures (e.g., Toeplitz).

The TDLMS filter is a flexible tool in that it does not attempt to change the working principles and the architecture of LMS filter. Therefore, the transform-domain module can precede other algorithms, such as RLS, GAL, and LSL. Notice that the conventional unitary transformations are independent of the underlying data, hence not optimal in regularizing condition numbers of the autocorrelation matrices of arbitrary real-time datasets. As an example, the DCT has been shown to be near-optimal for Toeplitz matrices. However, the DCT loses its near-optimality in conditioning sparse linear systems.

From a different perspective, the transformation of a matrix, such as autocorrelation matrix to improve the condition number is regarded as a subproblem of preconditioning of matrices [22]–[24]. Jacobi [25], [26], Gauss–Seidel (GS) [27], approximate inverse [28], [29], and incomplete lower-upper (LU) factorization [30] preconditioners are examples of such data-dependent transformations that utilize a decomposition of the input matrix consisting of coefficients of linear equations. These algorithms are well suited for solving linear system of equations. In short, the transformation that improves the performance of the LMS filter can also be applied to solving linear systems of equations.

However, not all the strategies for solving linear systems of equations using matrix-preconditioner are suitable for TDLMS. With careful attention, it can be seen that there is a difference between TDLMS and linear systems in terms of the usage of a preconditioner, as explained below. The preconditioning action is *implicit* in TDLMS filters (split preconditioner). The autocorrelation matrix is not explicitly used in the LMS architecture. Instead, it is the input data or the transformed input data that are channeled through the LMS lattice and the update of weights is based on error-correcting learning. While expressing the mse at the output as a function of filter weights, it appears that the transformation matrix at the input is attempting to condition the input autocorrelation matrix and the convergence of the LMS algorithm depends on the input autocorrelation matrix. On the other hand, in case of solving linear system of equations, ($Ax = b$) type, the use of a preconditioner is *explicit*, which is $M^{-1}Ax = M^{-1}b$, with M as a preconditioner matrix, such as the GS type.

Let A be a matrix to be preconditioned by another matrix ζ. ζ is said to be a left, right, and split preconditioner if $\zeta^{-1}A$, $A\zeta^{-1}$, and $U_1^{-1}AU_2^{-T}$ with $\zeta = U_1U_2^T$, respectively, provide improved condition numbers compared to that of A. GS, incomplete LU, and approximate inverse are examples of a left preconditioner. The transformations in the TDLMS algorithm are the unitary split preconditioner type. By the unitary property, we have ($U_1U_2^T = I$) and $U_1 = U_2 = U$. Therefore, it is the unitary split preconditioner matrix that can be applied to TDLMS as well as to solving linear systems of equations. In PrecoG, we aim to learn such unitary split preconditioner from input data. In addition, as the transformation is unitary, the energy of input data remains unchanged after transformation.

It is to note that our approach does not follow the traditional graph Laplacian regularization-based learning algorithms. For example, the work in [31] employed graph regularization [32], where feature vectors corresponding to each class are expected to be in proximity after regularization. The main objective of such works is the improvement of classification scores. In our work, the condition number of the autocorrelation matrix of the transformed data is expected to approach unity after learning the input graph. Our objective is to speed up the convergence of the LMS filter weights.

Unlike the off-the-shelf, data-independent transformations in TDLMS, the derivation of the PrecoG unitary split conditioner is motivated by the topology of the structured input data. The topology determines neighborhood relationship between data points, which can be represented using graph-theoretic tools [33]–[35]. In recent years, manifold processing and regularization have shown promise in various applications [36], [37]. Based on such evidence, we hypothesize that the intrinsic topology of the input data affects the construction of a suitable preconditioner matrix. We estimate a data manifold that provides a set of orthonormal bases acting as a split preconditioning matrix. The data, when projected onto the basis, are expected to be decorrelated. In fact, learning paradigms that use the gradient descent approach as an intermediate step to update model weights as a function of online data will be substantially benefited from our work.

The *main contributions* of this work are as follows.

1) Unlike the traditional off-the-shelf transformations, PrecoG provides an *optimization* framework that recursively finds the desired unitary transformation for the preconditioning matrix. The optimization leverages the estimation of the data manifold using a graph.
2) We show that our approach is equally applicable in preconditioning arbitrary linear systems apart from ameliorating the convergence of LMS filters.
3) Another advantage of PrecoG is that it can be applied without having a prior knowledge about the process that generates the input data. The estimated autocorrelation matrix serves our purpose.

To our knowledge, no mathematical framework has been presented so far that could potentially generate such matrices from the data.

A. Why a Graph Theoretic Approach?

For an N-dimensional data, there are N taps in the classical LMS filter (see Appendices; LMS and TDLMS algorithms) and these taps are independent of each other. Instead, in our model, we assume that there is a relationship among the taps that evolves as the data stream passes. The relationship can be encoded using a graph with edge weights that are to be estimated. By way of the theory of graph signal processing, the Laplacian matrix provides a set of orthonormal bases for the data. The set of bases acts as a preconditioning matrix.

In our brain, the neurons altogether form a complex network [38]–[40]. The classical LMS filter bears a weak analogy with brain neuronal networks, where each tap acts as a neuron. The neurons "innervate" postsynaptically (considering only monosynaptic connections) another neuron, which acts as a summing node or an integrator [see Fig. 1(C)]. Each weight mimics the synaptic strength. The weight update step utilizes the LMS algorithm, which is error-correcting in nature by its formulation. We assume that the input graph (barring the integrator or summing neuron) to be *fully-connected*, undirected, weighted graph [see Fig. 1(D)]. Using the transformation, a tap (or equivalently a neuron) can exert positive (excitatory) or negative (inhibitory) [41] effect on its neighbor, including itself to precondition the input data as shown in Fig. 1(D).

II. LMS AND TDLMS ALGORITHMS (IN BRIEF)

Let X be a real-valued data that are channeled to a filter with N tap delays, $\Phi(t) = [\phi_0(t), \phi_1(t), \ldots, \phi_{N-1}(t)]$. The data vector at time $t = n$ at the taps can be written as

$x_n = [x(n-N+1), x(n-N+2), \ldots, x(n)]$. The filter output is given by $y(n) = \Phi(n)^T x_n$. Let the desired response be $d(n)$. Then, the error $e(n) = d(n) - y(n) = d(n) - \Phi(n)^T x_n$.

The LMS algorithm is formulated as follows:

$$\begin{aligned}\phi_i(n+1) &= \phi_i(n) - \mu \frac{\partial e^2(n)}{\partial \phi_i(n)} \\ &= \phi_i(n) - 2\mu e(n) \frac{\partial \big[d(n) - \Phi(n)^T x_n\big]}{\partial \phi_i(n)} \\ &= \phi_i(n) + 2\mu e(n) x_{(n-i)}. \end{aligned} \tag{1}$$

Assume that $\phi(n)$ is independent of X(n) at the convergence (because when $\Phi(n)$ converges to a stationary array, X(n) may keep changing). Taking expectation on both sides of the above expression, we get

$$\begin{aligned}E(\Phi(n+1)) &= E(\Phi(n)) + 2\mu E(e(n)x_n) \\ &= E(\Phi(n)) + 2\mu E\big(x_n\big(d(n) - x_n^T \Phi(n)\big)\big) \\ &= E(\Phi(n)) + 2\mu E(x_n d(n)) \\ &\quad -2\mu E\big(x_n x_n^T\big) E(\Phi(n)) \\ &= E(\Phi(n)) + 2\mu R_{dx} - 2\mu R_{xx} E(\Phi(n)) \\ &= (1 - 2\mu R_{xx}) E(\Phi(n)) + 2\mu R_{dx}. \end{aligned} \tag{2}$$

Let us consider the term $E(\Phi(n))(1-2\mu R_{xx})$. Let us assume $R_{xx} = R$ and $R_{xx} = U \Lambda U^T$. Therefore,

$$\begin{aligned}E(\Phi(n))(1 - 2\mu R_{xx}) &= U(I - 2\mu\Lambda) U^T E(\Phi(n)) \\ &= U(I - 2\mu\Lambda)^n U^T E(\Phi(0)). \end{aligned} \tag{3}$$

To converge $E(W(n+1))$ to a stable value $2\mu R_{dx}$ in (2), $(I - 2\mu\Lambda)^n$ must converge to zero at $n \to \infty$. This is possible only when $(1 - 2\mu\lambda_i) \to 0 \forall i$. So, the convergence will be quickly achieved if $(\lambda_{\max}/\lambda_{\min}) \to 1$.

In TDLMS, let the transformation be $T \in \mathcal{R}^{N \times N}$ and the transformed data vector is $x'_N = T x_N$. The following step is the normalization of power:

$$\begin{aligned}v_i(n) &= \frac{x'_{i(n)}}{\sqrt{P_i(n) + \text{eps}}}; \quad \text{eps} = \text{small number} \\ P_i(n) &= \beta P_i(n-1) + (1-\beta)\big(x'_{i(n)}\big)^2. \end{aligned} \tag{4}$$

Next, $v(n)$ is channeled to the LMS filter. The expression can be obtained simply by replacing $x_{(n-i)}$ with $v(n-i)$ in (1).

III. Graph Theory (in Brief)

A graph can be compactly represented by a triplet $(\mathcal{V}, \mathcal{E}, \boldsymbol{w})$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ the set of edges, and $\boldsymbol{w}$ the weights of the edges. For a *finite* graph, $|\mathcal{V}| = N$ which is a finite positive integer, and $|\bullet|$ is the cardinality of a set. By denoting $w_{ij} \in \boldsymbol{w}$ as a real non-negative weight between two vertices i and j with $i, j \in \{1, 2, \ldots, N\}$, the adjacency matrix, A of $\mathcal{G}$ can be given by $a_{ij} = w_{ij}$ with $a_{ii} = 0$ for a graph with no self-loop. $A \in \mathcal{R}^{N \times N}$ is symmetric for an undirected graph and can be sparse based on the number of edges. The incidence matrix, $B \in \mathcal{R}^{N \times |\mathcal{E}|}$ of $\mathcal{G}$ is defined as $b_{ij} = 1$ or -1 where the edge j is incident to or emergent from the vertex i. Otherwise $b_{ij} = 0$. The graph Laplacian $L \in \mathcal{R}^{N \times N}$, which is a symmetric positive-semidefinite matrix, can be given by $L = BWB^T$, where W is a diagonal matrix containing $\boldsymbol{w}$. Determining the topology of input data refers to the estimation of A or L depending on the formulation of the problem at hand.

IV. Problem Statement

Let $x_k = [x(k) \; x(k-1) \; \cdots \; x(k-N+1)]$ be an N-length real-valued tap-delayed input signal vector at kth instant. The vector representation is convenient for estimating the input autocorrelation as an ergodic process. Let the autocorrelation matrix, denoted by R_N be defined as $R_N = E(x_N x_N^T)$. We assume that Δ_Y is the main diagonal of a square matrix Y ($\Delta_Y = \text{diag}(Y)$). Following this notation, after power normalization the autocorrelation matrix becomes $S_N = \Delta_{R_N}^{-1/2} R_N \Delta_{R_N}^{-1/2}$. In general, the condition number of S_N, χ_{S_N}, happens to be significantly large in practical datasets. For example, the condition number of the autocorrelation matrix of a Markov process with signal correlation factor as 0.95 has a χ of $\mathcal{O}(10^3)$. Notice that we seek U_N to minimize the condition number of S_N. Let a unitary transformation be U_N ($U_N U_N^T = I$) such that the transformed autocorrelation matrix becomes $\tilde{R}_N = E[U_N^T x_k x_k^T U_N]$. Next, $\tilde{R}_N$ is subjected to a power normalization stage that produces $\tilde{S}_N = \Delta_{\tilde{R}_N}^{-1/2} \tilde{R}_N \Delta_{\tilde{R}_N}^{-1/2}$. Precisely, we want the eigenvalues of $\lim_{N\to\infty} \tilde{S}_N \in [1 - \epsilon_2, \; 1 + \epsilon_1]$, where ϵ_1 and ϵ_2 are arbitrary constants such that $\chi_{\max} \simeq ((1+\epsilon_1)/(1-\epsilon_2))$. A schematic of our algorithm is given in Fig. 1.

Let us take an example of a first-order Markov input with the signal correlation factor ρ and autocorrelation matrix, R_N as

$$R_N = E\big[\boldsymbol{x}_k \boldsymbol{x}_k^H\big] = \begin{pmatrix} 1 & \rho & \cdots & \rho^{n-1} \\ \rho & 1 & \cdots & \rho^{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho^{n-1} & \rho^{n-2} & \cdots & 1 \end{pmatrix}. \tag{5}$$

It is shown in [21] that $\chi_{S_N} \simeq ((1+\rho)/(1-\rho))^2$, which suggests that $\epsilon_1 = \rho^2 + 2\rho$, and $\epsilon_2 = 2\rho - \rho^2$. After applying the DFT, the condition number becomes $\lim_{N\to\infty} \chi_{\tilde{S}_N} = ((1+\rho)/(1-\rho))$, which indicates that $\epsilon_1 = \epsilon_2 = \rho$. On applying the DCT, $\lim_{N\to\infty} \chi_{\tilde{S}_N} = 1 + \rho$ with $\epsilon_1 = \rho$ and $\epsilon_2 = 0$.

The optimal convergence properties are obtained when $\tilde{S}_N$ converges to the identity matrix in the *rank zero perturbation* sense [21]: A and B with $\eta = A - B$ have the same asymptotic eigenvalue distribution if

$$\lim_{N \longrightarrow \infty} \text{rank}(\eta) = 0. \tag{6}$$

In our case, with λ as an eigenvalue, this translates to

$$\lim_{N\to\infty} \det\big(\tilde{S}_N - \lambda \mathbb{I}_N\big) = 0 \tag{7}$$

which can be expanded as

$$\begin{aligned}\lim_{N\to\infty} \det\Big(\Delta_{\tilde{R}_N}^{-1/2} \tilde{R}_N \Delta_{\tilde{R}_N}^{-1/2} - \lambda \mathbb{I}_N\Big) = 0 \\ \lim_{N\to\infty} \det\big(\tilde{R}_N - \lambda \Delta_{\tilde{R}_N}\big) = 0. \end{aligned} \tag{8}$$

However, in the presence of the determinant in (8), obtaining a closed-form expression for U_N is difficult to obtain. To overcome this obstacle, we apply the Frobenius norm in (9). The modified optimization problem becomes

$$U_N = \underset{U_N \in O(N)}{\operatorname{argmin}} \left\| \tilde{R}_N - \lambda \Delta_{\tilde{R}_N} \right\|_F^2. \tag{9}$$

Here, $O(N)$ is the set of unitary matrices. The Frobenius norm imposes a stronger constraint compared to (7). In fact, while (8) can be solved if at least one column of $\tilde{R}_N - \lambda \Delta_{\tilde{R}_N}$ can be expressed as a linear combination of rest of the columns, (9) becomes zero only when $\tilde{R}_N - \lambda \Delta_{\tilde{R}_N}$ is a zero matrix. In effect, the Frobenius norm reduces the search space of U_N. It is due to the fact that the set of U_N that solves (9) is a subset of the U_N that are also the solutions of (8).

Here, we address two aspects of the problem. First, (9) attempts to minimize the difference between $\tilde{R}_N$, which is $U_N^T R_N U_N$, and the scaled diagonal matrix of $\tilde{R}_N$. This is necessary because it accounts for the spectral leakage [21] as detailed later in this section. The second aspect is that (9) attempts to diagonalize $\tilde{R}_N$ apart from attempting to make the eigenvalues unity only. We relax the unity constraint by forcing the eigenvalues to lie within a range $[1-\epsilon_2,\ 1+\epsilon_1]$, expecting the condition number to approximate $((1+\epsilon_1)/(1-\epsilon_2))$.

The question is: how to obtain such U_N from the input data? We consider the N-tap LMS filter as a graph having N nodes. The graph is fully connected and weighted [see Fig. 1(C)]. Using this model, it can be seen that U_N is a function of edge weights, $\boldsymbol{w}$ (see Section III). It is to remind the readers that $\boldsymbol{w}$ is not the set of weights of LMS filter taps. Instead, $\boldsymbol{w}$ represents the set of edge weights of the input graph that we modeled. This fictitious graph will provide U_N. Next, this U_N will be applied to the input data. The transformed data will be channeled to the LMS filter. The filter tap weights will then be recursively updated using the LMS rule to adapt the transformed data.

Incorporating the above discussion, the optimization problem with parameters $p = (\boldsymbol{w}, \epsilon_1, \epsilon_2)$ becomes

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left\{ \underbrace{\| \tilde{R}_N - s_+ \Delta_{\tilde{R}_N} \|_F^2 + \| \tilde{R}_N - s_- \Delta_{\tilde{R}_N} \|_F^2}_{E(p)} + \beta\left(\boldsymbol{w}^T \boldsymbol{w} - 1\right) \right\} \tag{10}$$

where $s_+ = 1 + \epsilon_1$ and $s_- = 1 - \epsilon_2$ are the upper and lower bounds, respectively, for the eigenvalues of S_N. To prevent each w_i from erratic values during iteration, we impose 2-norm on the weight vector $\boldsymbol{w}$.

The cost function in (10) is nonconvex. Therefore, the solution is not guaranteed to be a global optimum. In our work, the required solution is obtained through gradient descent with μ as the step size parameter

$$w_i^{t+1} = w_i^t(1-2\beta) - \mu \operatorname{Tr}\left(\left[\frac{\partial E(p)}{\partial U_N} \right]^T \frac{\partial U_N}{\partial w_i} \right); \quad 0 < \mu < 1. \tag{11}$$

t is the iteration index, and the computation of $(\partial E(p)/\partial U_N)$ is given in Appendix B.

V. Methodology

A. Laplacian Parametrization

The problem of finding a suboptimal transform by optimizing (10) is solved by leveraging the graph framework. In this framework, the input data are modeled with a finite, single-connected, and undirected graph without self-loops and is endowed with a set of vertices, edges, and edge weights. For example, for an LMS filter with N taps, an input signal vector x_k has length N, which can be represented with N vertices. Basically, each vertex corresponds to one tap of the LMS filter. Using the graph, the unknowns of the optimization in (10) are the number of edges and the associated edge weights. A fully-connected graph with N vertices contains $((N(N-1))/2)$ edges. A deleted edge can be represented with zero edge weight. We denote the set of unknown parameters as $\boldsymbol{w}$, which is the set of nonzero weights of the graph.

To find the desired transformation U_N, the algorithm initializes the weights $\boldsymbol{w}$ with random numbers sampled from a Gaussian distribution with zero mean and unit variance. Let W is the diagonal matrix containing $\boldsymbol{w}$. Then by definition, the graph Laplacian, which is symmetric and positive semidefinite by construction, is given by $L = BWB^T$. B is the incidence matrix as mentioned in Section III. The spectral decomposition of L provides the matrix of eigenvectors U. Finally, $U^T x_k$ is the transformation that is expected to decorrelate the dataset, which may not be possible due to random initialization. Then the cost function (10) helps update the weights and the search for the desired transformation continues in an iterative fashion until the objective conditions are met.

As the cost function is nonconvex, the performance of PrecoG depends on how the graph adjacency matrix is initialized. For all our experiments, we generated a matrix, S, for which the entries are sampled from a random Gaussian (zero mean, unit variance) distribution. Later, we performed the following operation, $S \longleftarrow |(S + S')/2|$, where $'$ and $|\dot{|}$ are the matrix transpose and elementwise absolute value operations, respectively. This is followed by $S \longleftarrow S - \operatorname{diag}(S)$ to delete the diagonal entries of S to make the graph simple. The absolute value operation is needed to make the edge weights non-negative. Since the graph is fully connected, the incidence matrix B is known beforehand [see Fig. 1(A)]. This incidence matrix B is applied to (14).

B. Computation of Partial Derivatives

From Section III, we obtain that $L = BWB^T$. Let $\Theta_i = (\partial L/\partial w_i)$, which can be evaluated as $(\partial L/\partial w_i) = B(\partial W/\partial w_i)B^T$. In (11), the computation of $(\partial U_N/\partial w_i)$ is performed by

$$\frac{\partial u_{k,l}}{\partial w_i} = \operatorname{Tr}\left(\frac{\partial u_{k,l}}{\partial L} \Theta_i \right) = \operatorname{Tr}\left(\left[\frac{\partial L}{\partial u_{k,l}} \right]^{-T} \Theta_i \right) \tag{12}$$

where using $J^{mn} = \delta_{mk}\delta_{nl}$, $(\partial L/\partial u_{kl})$ can be given by

$$L = U_N \Gamma U_N^T \Longrightarrow \frac{\partial L}{\partial u_{kl}} = U_N \Gamma J^{mn} + J^{nm} \Gamma U_N^T. \tag{13}$$

However, it is difficult to compute (11) because $(\partial L/\partial u_{kl})$ is non-invertible. The proof is given in Appendix C. One can use diagonal compensation by adding $\alpha I (\alpha < 0.001)$ to $(\partial L/\partial u_{kl})$ [42]. However, it produces erroneous solution.

We approach the problem of computing $(\partial U_N/\partial w_i)$ using first-order eigenvector perturbation (see Appendix D)

$$\frac{\partial \boldsymbol{u}_i}{\partial w_j} = \sum_{\substack{p\neq i \\ \lambda_p \neq \lambda_i}} \frac{1-\delta(\lambda_i,\lambda_p)}{(\lambda_i-\lambda_p)} \left\langle \boldsymbol{u}_i, B\frac{\partial W}{\partial w_j}B^T\boldsymbol{u}_p \right\rangle \boldsymbol{u}_p + \sum_{\substack{q\neq i \\ \lambda_q=\lambda_i\neq 0}} \frac{\delta(\lambda_i,\lambda_q)}{\lambda_i} \left\langle \boldsymbol{u}_i, B\frac{\partial W}{\partial w_j}B^T\boldsymbol{u}_q \right\rangle \boldsymbol{u}_i + \sum_{\substack{q\neq i \\ \lambda_q=\lambda_i=0}} \delta(\lambda_i,\lambda_q) \left\langle \boldsymbol{u}_i, B\frac{\partial W}{\partial w_j}B^T\boldsymbol{u}_q \right\rangle \boldsymbol{u}_i. \quad (14)$$

The above formulation can be plugged in $(\partial U_N/\partial w_j) = [(\partial \boldsymbol{u}_1/\partial w_j)\ (\partial \boldsymbol{u}_2/\partial w_j)\ \cdots\ (\partial \boldsymbol{u}_N/\partial w_j)]$ to be used in (11). In (14), the space of incremental changes in an eigenvector with respect to edge weight w_i is a spanned by the eigenvectors $\boldsymbol{u}_i$.

The data-dependent transformation matrix, U utilizes the autocorrelation matrix of the input data [$\tilde{R}_N$ inside $E(p)$].

VI. Results

We split the result section in three parts. The first part considers systems where the matrix that is to be conditioned is known *a priori*. The second presents the performance of PrecoG on simulated on-line data, where autocorrelation matrices are estimated using a window. Finally, the third presents the complexity of PrecoG computation.

Part-I (Linear Systems): We show the effectiveness of our approach in preconditioning different matrices against the preconditioners—DCT, DFT, Jacobi (tridiagonal matrix type), GS, and incomplete LU factorization. To represent the strength of an individual algorithm, we incorporate the condition number of each unconditional matrix with the aforementioned methods. To scale the condition numbers obtained from several methods with respect to ours, we define a metric, condition ratio = (condition number obtained from a method)/(condition number obtained by PrecoG). In some of the results, we compute $\log_{10}$(condition ratio) to mitigate the enormous variance present in the condition ratio scores.

First, we apply our method to precondition a Hilbert matrix [43] which is severely ill-conditioned. Hilbert matrix, H is defined as $H(i,j) = (1/(i+j-1))$. In the experiment, we add a regularizer using $(\alpha\mathcal{I})$ with $0 < \alpha \leq 1$ as the regularization coefficient. The condition ratios of the existing algorithms, including PrecoG on preconditioning the Hilbert matrices, which are regularized by changing the α, are shown in Fig. 1(E-a). Notice that the X-axis is given in $-\log_{10}$ scale. Therefore, smaller values at X coordinate indicates higher regularization of the Hilbert matrix. On decreasing the value of α, the Hilbert matrix becomes severely ill-conditioned, and the performance of the competitive algorithms except GS exhibit inconsistent behavior. The DCT performs better near $\alpha = 1$ $(-\log10(\alpha) = 0)$ because of the diagonally dominant nature of the matrix. PrecoG outperforms all the comparative methods.

We also evaluate our algorithm on five different random positive definite matrices with the values taken from a zero-mean and unit-variance Gaussian process. Here, we regularize the matrices to ensure positive-definiteness. It is evident from Fig. 1(E-b), the condition ratios obtained by applying PrecoG outperforms the DCT, GS, DFT, and Jacobi transformations.

The condition ratios (in log10 scale) with respect to PrecoG on sparse systems of equations are shown in Fig. 1(E-c). For PrecoG, log10 of the condition ratio is zero. So, it is not shown in the plot. The three sparse matrices are random by construction with sparsity (number of nonzero elements/total number of elements) levels as $[1/4, 2/7, 1/5]$, respectively. PrecoG significantly outperforms the conventional transformations.

Part-II (Simulated Process): PrecoG is evaluated on two simulated datasets from two different processes—first-order Markov process and second-order autoregressive (AR) process. The LMS algorithm is supervised on these datasets in a sense that the desired output is known beforehand for each dataset. The data are channeled to the filter in on-line mode, and PrecoG is blinded against the customization of the data. The autocorrelation matrices of both processes are Toeplitz. So far, the DCT is known as the near-optimal preconditioner for first- and second-order Markov processes. We have also considered the performance of PrecoG on unsupervised Hebbian-LMS learning [41].

A. First-Order Markov or AR(1) Process

We simulate a first-order AR process with a set of signal correlation factors, ρ. As mentioned in the third claim of our main contributions, PrecoG is shown to performed significantly well without the prior knowledge of asymptotic autocorrelation matrix of the input data.

For each ρ, we have simulated 2000 samples of 1-D data and convolved the data with a filter defined by coefficients $h = [1\ \ -0.8\ \ 6\ \ 3]$ to generate the desired output. The data power is taken as unity. The output is added with a white Gaussian noise of signal power 0.01. The goal is to converge the filter weight to the impulse function of the convolution filter. Conventional transformations, such as the DCT and the DST are applied at the beginning. However, due to data-dependent nature of PrecoG, we consider a part of initial data to estimate the time-averaged autocorrelation function. The final unitary preconditioner of PrecoG depends on the length of the initial data. The step length of the LMS algorithm is set as 0.002. For a fixed length of initial data, we have created a search space of $L2$ coefficients and PrecoG learning rate, to find out the transformation.

It is evident from Fig. 1(F) that the number of taps in the LMS filter is 4. We have investigated the relationship between these four taps (neurons) by varying the initial data window. Here, $R-est$ 25 means that the estimated autocorrelation is averaged over the first 25 data and the "gray"-scaled matrix is the PrecoG transformation U_N. The grayscale describes the strength of excitatory (toward white) and inhibitory (toward

TABLE I

TABLE LISTS THE CONDITION NUMBERS THAT ARE OBTAINED BY THE DCT AND PRECOG TRANSFORMATIONS ON AR(1) DATA WITH $\rho = 0.95$. WITH DIFFERENT INITIAL WINDOW LENGTHS, THE ESTIMATED AUTOCORRELATION MATRICES (R_{est}) VARY. THE SECOND AND THE FOURTH COLUMNS SHOW THE CONDITION NUMBERS BY APPLYING PRECOG AND DCT ON THE ESTIMATED R, RESPECTIVELY. USING SIMPLE DERIVATIONS, THE ASYMPTOTIC R_{asymp} IS GIVEN IN (5). THE EFFECT OF PRECOG AND THE DCT ARE GIVEN IN THE THIRD AND FIFTH COLUMNS, RESPECTIVELY. THIS TABLE SHOWS THAT WITH HIGH SIGNAL CORRELATION FACTOR, SUCH AS $\rho = 0.95$, WE NEED AT LEAST 75 LENGTH INITIAL WINDOW TO ESTIMATE R, WHEN THE CONDITION NUMBER BY PRECOG DROPS JUST BELOW THE DCT WHEN APPLIED ON THE ASYMPTOTIC R

Data length (ρ=0.95)	χ_{PrecoG} on R_{est}	χ_{PrecoG} on R_{asymp}	χ_{DCT} on R_{est}	χ_{DCT} on R_{asymp}
25	1.0094	1.3550	1.2390	1.15
50	1.0071	1.3404	1.2609	1.15
75	1.0100	1.1251	1.2212	1.15
100	1.0348	1.0544	1.1142	1.15
125	1.0232	1.0188	1.0998	1.15
150	1.0141	1.0395	1.1178	1.15
175	1.0130	1.0418	1.1162	1.15
200	1.0054	1.0306	1.1146	1.15
225	1.0253	1.0489	1.0937	1.15
250	1.0246	1.0448	1.0949	1.15
275	1.0052	1.0525	1.1135	1.15
300	1.0222	1.0611	1.1230	1.15
325	1.0016	1.06	1.1025	1.15
350	1.0012	1.0453	1.1090	1.15
375	1.0144	1.0502	1.1111	1.15
400	1.0144	1.0536	1.1142	1.15

TABLE II

TABLE LISTS THE CONDITION NUMBERS THAT ARE OBTAINED BY THE DCT AND PRECOG TRANSFORMATIONS ON AR(1) DATA WITH $\rho = 0.2$. THIS TABLE SHOWS THAT WITH SMALL SIGNAL CORRELATION FACTOR, SUCH AS $\rho = 0.2$, WE DO NOT NEED TO HAVE A LONGER WINDOW TO ESTIMATE R. IN OUR SIMULATION, AT THE INITIAL WINDOW LENGTH OF 25, THE CONDITION NUMBER BY PRECOG DROPS JUST BELOW THE DCT WHEN APPLIED ON THE ASYMPTOTIC R. THIS MAKES SENSE AS THE DATA DOES NOT HAVE ENOUGH CORRELATION AMONG ADJACENT SAMPLES BECAUSE OF THE LOW CORRELATION FACTOR. SO, ANY PRECONDITIONING MATRIX WILL TAKE A RELATIVELY SHORT TIME TO FURTHER DECORRELATE R

Data length (ρ=0.2)	χ_{PrecoG} on R_{est}	χ_{PrecoG} on R_{asymp}	χ_{DCT} on R_{est}	χ_{DCT} on R_{asymp}
25	1.0013	1.0193	1.3368	1.089
50	1.0002	1.0098	1.4703	1.089
75	1.0015	1.0176	1.2846	1.089
100	1.0004	1.0056	1.3514	1.089
125	1.0005	1.0057	1.3555	1.089
150	1.0002	1.0092	1.4060	1.089
175	1.0004	1.0020	1.3336	1.089
200	1.0001	1.0060	1.3422	1.089
225	1.0003	1.0093	1.3056	1.089
250	1.0005	1.0048	1.2627	1.089
275	1.0009	1.0028	1.2707	1.089
300	1.0002	1.0068	1.2780	1.089
325	1.0003	1.0095	1.2903	1.089
350	1.0002	1.0071	1.2541	1.089
375	1.0002	1.0083	1.2559	1.089
400	1.0003	1.0067	1.2274	1.089

black) drives. The relationship (matrix) is asymmetric. As an example to interpret this asymmetry, it can be seen that for $R - est$ 25, neuron 1 inhibits neuron 2, but neuron 2 applies excitatory postsynaptic potential to neuron 1. For example, neuron 1 in R-est 25 exerts an excitatory drive onto itself (R-est 25 (1,1) is above zero). Fig. 1(G) exhibits the transformation of the DCT, the DST, and PrecoG, where the PrecoG transformation is obtained from a different instance of AR(1) process with same h.

A comparison between the DCT and PrecoG in terms of the convergence of filter weights is given in Fig. 1(H). The DCT is applied to the input data prior to resuming the LMS filter operation. PrecoG functions in two stages. At first, PrecoG computes the transformation using an initial data window (in this example, window length is 200), during which the filter taps are updated without applying any transformation to the input. Once the transformation is obtained, it is applied to the rest of the data stream (TDLMS). It can be verified from Fig. 1(F) and (G) that with DCT, the tap weight takes a longer time to converge compared with PrecoG. The learning rate for LMS is static for both of these processes.

A relevant question is: how long does the initial data window have to be to compute an effective preconditioner? Table I in Fig. 1 and Table II in Fig. 2 provide an answer to that question. In each table, we provide results for a set of attributes. First, DCT and PrecoG are tested regarding how they perform on the estimated time-averaged R (R_{est}) and asymptotic R [see (5)]. Let us take the first row of Table I. With the initial data window of length 25, we have obtained a transformation U. The DCT gives the condition number of 1.2390 on R_{est}, where applying PrecoG yields the condition number of 1.0094 on R_{est}. It might appear that PrecoG performed well over the DCT if the initial 25 data samples are considered. However, the third column reveals that with U (PrecoG) at hand, it yields the condition number of 1.3550 when the asymptotic R is considered and it is inferior to the DCT (1.15). This due to the fact that R_{est} is not a robust approximation of the asymptotic R. It is not until the data window of length 75 that we can observe that PrecoG (1.1251) starts performing better than the DCT (1.15) on the asymptotic R.

Tables I and II are the results on AR(1) process at two different signal correlation factors 0.95 and 0.2, respectively. One can see that PrecoG starts performing better than DCT with the data window length 25 in case of $\rho = 0.2$. This

observation is harmonious with the theory of TDLMS. AR(1) with $\rho = 0.95$ is a highly correlated datastream. Therefore, a longer data window is required to estimate R to decorrelate the data. Fig. 1(H-b) shows the transformation matrices at three different initial length of data window for AR(1) process with $\rho = 0.2$. When compared to Fig. 1(F) (AR(1) with $\rho = 0.95$), it is evident that with smaller size data window, PrecoG contains a large number of inhibitory drives when the signal correlation factor is relatively low.

B. Second-Order Autoregressive Process

In Fig. 2(a), we present the condition ratios computed by applying the algorithms on the autocorrelation matrices of eight second-order AR process with parameters (ρ_1, ρ_2) [16].

The input autocorrelation matrix R_N of such process is given by $R_N = c_1 R_N(\rho_1) + c_2 R_N(\rho_2)$. $R_N(\rho_1)$ and $R_N(\rho_2)$ are two Toeplitz matrices, similar to R_N of first-order Markov process. c_1 and c_2 are constants and are given by $c_1 = ((\rho_1(1-\rho_2^2))/((\rho_1-\rho_2)(1+\rho_1\rho_2)))$, $c_2 = ((-\rho_2(1-\rho_1^2))/((\rho_1-\rho_2)(1+\rho_1\rho_2)))$. As shown in Fig. 2(a), PrecoG outperforms DCT in all the above cases, implying that it has better decorrelating ability than the DCT.

Next, we apply PrecoG to a simulated 3200-length data sample from an AR(2) process with $\rho_1 = 0.6$ and $\rho_2 = 0.9$. The AR(2) data power is set to unity. We assume that there are three filter taps. The convolution filter $h = [1\ 0.8\ -3]$ to generate the desired output. The convolved response is corrupted with additive white Gaussian noise with normalized signal power of 0.01. For each data x, we set LMS learning step as 0.001, and the power normalization factor $beta$ as 0.85. The length of initial data window to estimate the autocorrelation matrix is set as 200. Fig. 1(I-a) shows the behavior of a filter tap in the cases of DCT-LMS and PrecoG-LMS where each data sample is processed. The tap weight attains convergence faster in PrecoG accompanied by an accelerated reduction in error [see Fig. 1(I-b)] when compared with the DCT.

C. Hebbian Learning and Hebb-LMS Algorithm

In 1949, Hebb [44] postulated that concurrent synaptic changes occur as a function of pre and postsynaptic activity, which was elegantly stated as "neurons that are wired together fire together." This classical remark of Hebb attempted to explain the plausible role of synaptic changes in learning and memory. Hebbian and LMS were predominantly regarded as two distinct forms of learning. Hebbian form of learning is unsupervised in nature, whereas LMS is primarily supervised. However, Hebbian rule, when translated into computational filters, leads to instability. Neuroscience researchers explained that neurons and the network collectively maintain stability by scaling the data at the input synapses (homeostatic plasticity) [45], [46]. From the computational point of view, a parallel of this Hebbian learning and synaptic scaling with our work is given in Appendix A (Hebb-LMS algorithm). Widrow *et al.* [41], inventor of LMS filters, proposed Hebb-LMS algorithm in 2019. It describes a mechanism of learning in equilibrium (analog of homeostatic learning) using

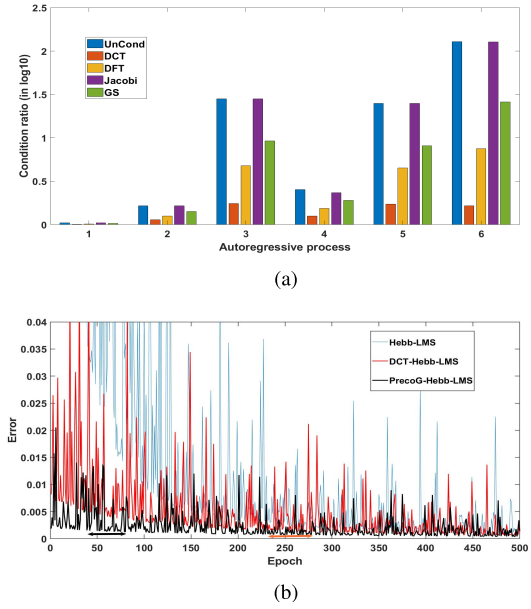

Fig. 2. (a) Plot shows comparative performances of DCT, DFT, Jacobi, and GS with respect to PrecoG (condition ratio in log10) on six AR(2) datastreams with (ρ_1, ρ_2) as $(0.15, 0.1)$, $(0.75, 0.7)$, $(0.25, 0.01)$, $(0.75, 0.1)$, $(0.9, 0.01)$, and $(0.99, 0.7)$. (b) Error profile over epochs in case of Hebb-LMS, and the effect of PrecoG and the DCT on the Hebb-LMS process. With PrecoG, the error is reduced significantly within 60 epochs.

a sigmoid. PrecoG performs efficaciously in comparison using transformed domain Hebb-LMS algorithm.

The relevance of this Hebbian learning and homeostatic rule (synaptic scaling) in our work can be speculated from a computational point of view. The error at the output corresponding to an input data is backpropagated and eventually updates the weights. A data vector is an array of real numbers. The positive entries in a vector collectively constitute an excitatory drive and the negative entries constitute an inhibitory drive to the postsynaptic neuron, where the drives are summed and later nonlinearly (such as, a sigmoid function) transformed. An ideal transformation (TDLMS) should have the property that it can regulate the magnitude of both the drives. A suitable transformation is expected to assess the balance of excitatory and inhibitory drives and modulate the input to achieve decorrelation. We resort to the input autocorrelation matrix that may provide such assessment. It is to remind the reader that input data is transformed prior to passing to the filter. Therefore, the transformation "scales" the data for the LMS filter. So, it acts as a regularizer for the adaptation of filter weights. If this philosophy is true, PrecoG should also work in the unsupervised settings because finding the transformation is independent of how the filter weights are updated.

The experimental setting considers a 3200 length data generated by an AR(1) process with a Gaussian noise with zero mean and a variance of 3. We assume that there are four filter taps. For each data x, the desired output is $Sigmoid(W^T x)$. We set the $slpha$ of the sigmoid as 0.5, $gamma$ as 0.5, the LMS learning rate as 0.001, and power normalization factor $beta$ as 0.85. The length of initial data window to estimate the autocorrelation matrix is set as 100. We run 500 epochs

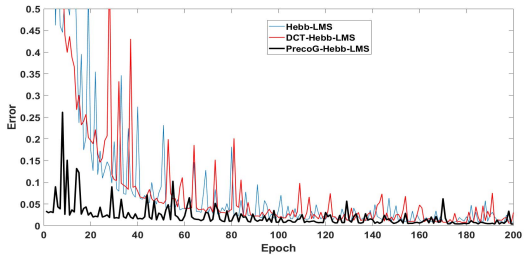

Fig. 3. Error profile over epochs in case of Hebb-LMS, and the effect of PrecoG and the DCT on the Hebb-LMS process on an AR(2) data. The data are shuffled at each epoch prior to passing to the filter. R is estimated from the shuffled in the first epoch. With PrecoG, the error is reduced significantly within 30 epochs.

TABLE III

COMPUTATION TIME IN SECONDS (MEAN ± STANDARD DEVIATION) OF PRECOG OBTAINED USING MATLAB. THE COMPUTATION TIME INCLUDES THE TIME REQUIRED TO FIND PRECOG IN THE SEARCH PACE. FOR $N = 16, 64, 128$, WE USED 4-REGULAR GRAPHS

N	3	5	16*	64*	128*
Weights	3	10	32	128	256
Computation time (s)	0.078 ± 0.01	0.39 ± 0.062	2.83 ± 0.12	86.61 ± 2.67	302.86 ± 14.42

to inspect the behavior of error. In each epoch, weights are continuously updated after each data sample is passed and the datastream is randomly shuffled. We compute the Euclidean norm of the 3200 errors in each epoch to plot. The setting is fixed for all Hebb-LMS, DCT-Hebb-LMS and PrecoG-Hebb-LMS methods. In TDLMS using PrecoG, the output error is reduced when compared to Hebb-LMS and DCT-Hebb-LMS. A similar conclusion can be drawn when PrecoG and DCT are tested on an AR(2) process with a Gaussian noise with zero mean and variance 5 (see Fig. 3).

Part-III (Complexity): The update of weights w_i requires the computation of three partial derivatives, which needs eigendecomposition of the Laplacian in every iteration. Eigendecomposition of a matrix has a computational complexity of $\mathcal{O}(N^3)$. Note that $B(\partial W/\partial w_j)B^T$ is fixed for w_j. By definition, $L = BWB^T$. So, $E = (\partial L/\partial w_j) = B(\partial W/\partial w_j)B^T$. E has exactly four nonzero entries—$E(k,k) = E(q,q) = 1$, $E(k,q) = E(q,k) = -1 \forall w_j = w_{kq}$. Therefore, $B(\partial W/\partial w_j)B^T\boldsymbol{u}_p$ requires constant computation. For example, if $w_j = w_{13}$, then $B(\partial W/\partial w_j)B^T\boldsymbol{u}_p = [u_{p1} - u_{p3}, 0, u_{p3} - u_{p1}, 0, 0, ..]^T$, having nonzero entries at the first and third locations. It implies that $\langle \boldsymbol{u}_i, B(\partial W/\partial w_j)B^T\boldsymbol{u}_q \rangle$ requires constant computation. Now, each $\boldsymbol{u}_i$ has length N. For each i, $(\partial \boldsymbol{u}_i/\partial w_j)$ requires $\mathcal{O}(N)$ time to add for all $(N-1)$ vectors. So, the time complexity to compute $(\partial U_N/\partial w_j)$ is $\mathcal{O}(N^2)$. For N taps and I iterations, the computational complexity becomes $\mathcal{O}(N^3 I)$. A large search space of the learning rate (11) and regularization (10) incurs additional computational load (see Table III).

It is to note that the constructions of preconditioners for solving linear systems by comparative methods, such as Jacobi ($\mathcal{O}(N^2)$), successive over-relaxation (SOR) ($\mathcal{O}(N^3)$), symmetric SOR ($\mathcal{O}(N^3)$), and GS ($\mathcal{O}(N^3)$) have faster associated run times compared to PrecoG. Here, complexity accounts for the inversion of each preconditioner matrix. However, the acceleration in the convergence of the LMS filter using PrecoG is also expected to partially compensate for the computational overload of PrecoG. This is shown in Table IV. For a large N, we used a sparse graph to avoid a large number of edge weights in a fully connected graph.

TABLE IV

COMPARISON OF THE NUMBER OF ITERATIONS AMONG PRECONDITIONING METHODS FOR SOLVING LINEAR EQUATIONS. A IS A HILBERT MATRIX ADDED WITH $\alpha\mathcal{I}$. b IS GENERATED FROM A GAUSSIAN DISTRIBUTION $G(0,1)$. WE USED GAUSS–SIEDEL LINEAR SOLVER (TOLERANCE $= 10^{-5}$). IN PRECOG, WE TAKE FULLY CONNECTED, SIMPLE GRAPHS FOR $N = 5, 11$ AND A 3-REGULAR GRAPH FOR $N = 256$ $(*)$

(N, α)	LMS	GS	Jacobi	ILU	PrecoG
(5, 0.001)	1292	343	1134	1212	41
(5, 0.1)	28	15	24	42	6
(11, 0.001)	2068	407	1765	2185	103
(256, 0.001)*	14503	5689	11940	6561	671

VII. DISCUSSION

LMS filters play pivotal roles in several applications, and convergence of the algorithm in terms of the filter weights, which are updated at each iteration, poses critical challenges to the quality of performance of LMS filters. TDLMS offers a solution to overcome the convergence issue by transforming the data prior to channeling it to the LMS filter. Conventional transforms serve as a palette of such transformation. However, the convergence of LMS is mediated by the shape of the real-life data manifold. If the shape of the data manifold is known *a priori*, only then we can robustly evaluate the efficacy of the transformations. This issue is largely overlooked in the existing literature. PrecoG provides a solution to this problem by presenting an optimization framework that intelligently encodes data manifold into the transformation matrix. The "data" are absorbed into the actual or estimated autocorrelation matrix. The edge weights of the input graph are varied over iterations till the condition number of the transformed autocorrelation matrix approaches unity. This is how each edge weight is an implicit function of the input data. Apart from symmetry by definition in case of real-valued data, autocorrelation matrices containing distinct structures, such as diagonally dominant, Toeplitz, Hankel, and circulant appear in special circumstances. For example, the autocorrelation matrix of first-order Markov process possesses Toeplitz structure. If the autocorrelation at two time points depends only on the time difference, then the autocorrelation matrix becomes Toeplitz. However, these assumptions are rarely valid in cases of data with dynamic changes over time. PrecoG evaluates the data manifold after each interval and accordingly finds the unitary preconditioner. Users can set a time interval after which the preconditioning matrix will be computed periodically to incorporate dynamic changes into the matrix. Nevertheless, this flexibility comes at the price of an additional computational overhead.

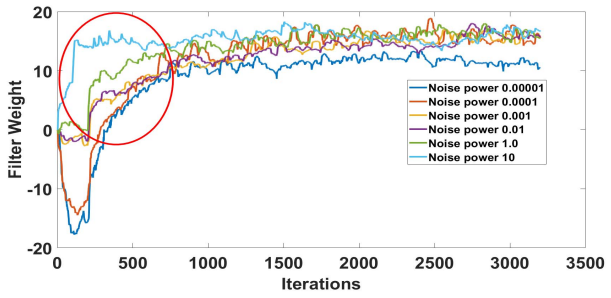

Fig. 4. Profiles of a tap of an LMS filter weight over iterations and at different noise power. An AR(0.9) process with unit signal power is transformed using PrecoG and is channeled through the four-tap LMS filter. Gaussian noise $\mathcal{N}(0,1)$ with varied power (σ_n) is added to the output. When the noise power is high, the input autocorrelation matrix approaches $R_N \sim \sigma_n^2 \mathcal{I}$, approximating unit condition number. The convergence becomes faster, as is evident in this plot. With lower noise power, the input becomes a highly correlated ($\rho = 0.9$) AR process. The difference in profiles of the tap weight over varying noise power can be observed in the encircled area.

PrecoG is shown to significantly outperform the conventional transformations that are considered in this article on AR datasets and different linear systems of equations. PrecoG is shown to have improved performance when the autocorrelation matrix of the process is estimated, making PrecoG amenable to be deployed in online scenarios. For AR datasets, we also examined the variation of the initial window length with the signal correlation factor. It is shown to be effective in unsupervised settings also, when we investigated PrecoG's performance in the Hebbian-LMS settings.

The coefficients Φ of a TDLMS filter are stable in mean and variance after applying the unitary transformation. In general, the step length μ affects the stability of Φ and a practical bound on μ can be given as $0 < \mu < (2/\text{tr}(R_N))$, where tr is the matrix trace operator. This bound ensures that $\phi \in \Phi$ will not diverge. In TDLMS, $\tilde{R}_N = U_N^T R_N U_N$. Now, it can be shown that $\text{tr}(A^T B) = \text{tr}(B A^T)$; $A, B \in \mathcal{R}^{m \times n}$. Using this result, it can be seen that $\text{tr}(U_N^T R_N U_N) = \text{tr}(U_N U_N^T R_N) = \text{tr}(R_N)$. Therefore, the stability of the TDLMS tap weights is not different from that of LMS if μ lies within the bound. With added noise power, the behavior of a filter tap of an LMS filter is shown in Fig. 4.

Practical applications, such as echo cancellation require a large number of filter taps (such as 512 or 1024) [3], [9]. In these applications, adaptive algorithms, such as RLS are not preferred because of large computational overhead. Algorithms, such as proportionate normalized LMS (PNLMS) and its variants are used in these applications. PrecoG has a large initial computational overhead for a large N. This overhead can be reduced in several ways: by finding a Bayesian framework for the search space to find the optimal learning rate and regularization parameter, by utilizing a sparse input graph structure, by way of eigendecomposition of a sparse matrix. We will explore these options in our future work.

VIII. Conclusion

In this work, we present a method to obtain a unitary split preconditioner by utilizing nonconvex optimization, graph theory and first-order perturbation theory. We demonstrate the efficacy of our approach over prevalent state-of-the-art techniques on Markov datasets and linear systems of equations. We inspect PrecoG in supervised and unsupervised settings. As a future endeavor, we will attempt to exploit the signal structure, input graph structure, and embed them into the optimization framework by including a set of constraints.

Appendix

A. Hebb-LMS Algorithm

In 1949, Hebb [44] postulated that concurrent synaptic changes occur as a function of pre and postsynaptic activity, which was elegantly stated as "neurons that are wired together fire together." This classical remark of Hebb attempted to explain the plausible role of synaptic changes in learning and memory.

So, Hebbian and LMS were predominantly regarded as two distinct forms of learning. Hebbian form of learning is unsupervised in nature, whereas LMS is primarily supervised. However, when Hebb's rule was applied to a linear filter to learn patterns in data by adaptively changing the weights, it posed a problem. If the concurrent pre and postsynaptic activities strengthen over time, there is nothing to scale the activity down. Researchers attempted to introduce a "forgetting" term inside the Hebb's rule. In 1982, Oja [47] introduced a modification to stabilize the computational analog of Hebb rule. In effect, the linear filter was able to learn the principal components of the input data. However, such modifications are *ad hoc* and, therefore, difficult to interpret.

Widrow *et al.* [41] proposed an algorithm that integrates Hebbian and LMS learning paradigms and it contains two equilibrium states for excitatory and inhibitory drives. The mathematical expression is given by

$$\begin{aligned} W(n+1) &= W(n) + 2\mu e(n) x(n) \\ e(n) &= \text{SGM}\big(W(n)^T x(n)\big) - \gamma\, W(n)^T x(n) \\ y(n) &= \begin{cases} \text{SGM}\big(W(n)^T x(n)\big), & \text{if } S > 0 \\ 0, & \text{o.w.} \end{cases} \end{aligned} \tag{15}$$

Here $S = \text{SGM}(W(n)^T x(n))$ and SGM is the polar sigmoid function. This algorithm is unsupervised because the target response of x(n) is S. It contains two stable equilibrium points 1 (excitatory) and -1 (inhibitory) on the SGM curve.

B. Evaluation of $[(\partial E(p)/\partial U_N)]$

Let, $M(\epsilon) = ||U^T R U - (1+\epsilon) U^T R U \circ \mathcal{I}||_F^2$. Then

$$\begin{aligned} M(\epsilon) = \text{Tr}\Big(&\big\{U^T R U - (1+\epsilon) U^T R U \circ \mathcal{I}\big\} \\ &\times \big\{U^T R U - (1+\epsilon) U^T R U \circ \mathcal{I}\big\}^T \Big). \end{aligned} \tag{16}$$

Equation (16) on expansion gives

$$\begin{aligned} M(\epsilon) = \text{Tr}\Big(&U^T R^2 U - 2(1+\epsilon)\big(U^T R U \circ \mathcal{I}\big) \\ &\times \big(U^T R U\big) + (1+\epsilon)^2 \big(U^T R U \circ \mathcal{I}\big)^2 \Big). \end{aligned} \tag{17}$$

Equation (17) is obtained using the fact that $UU^T = \mathcal{I}$. By performing the partial derivative

$$\begin{aligned}\frac{\partial M(\epsilon)}{\partial U} &= \frac{\partial \mathrm{Tr}\big(U^T R^2 U\big)}{\partial U} \\ &\quad - 2(1+\epsilon)\frac{\partial}{\partial U}\mathrm{Tr}\big\{\big(U^T RU \circ \mathcal{I}\big)\big(U^T RU\big)\big\} \\ &\quad + (1+\epsilon)^2\frac{\partial}{\partial U}\mathrm{Tr}\big(U^T RU \circ \mathcal{I}\big)^2. \\ &= 2R^2U - 4(1+\epsilon)RU + (1+\epsilon)^2\big(U^T RU \circ \mathcal{I}\big)RU.\end{aligned}$$

Next, $(\partial E(p)/\partial U)$ is computed using $(\partial M/\partial U)$ as $(\partial E(p)/\partial U) = ((\partial M(+\epsilon_1))/\partial U) + ((\partial M(-\epsilon_2))/\partial U)$

$$\begin{aligned}\frac{\partial E(p)}{\partial U_n} = 2\big[&2R_n - 2(2-\epsilon_1-\epsilon_2)\mathcal{I} \\ &-\big\{(1+\epsilon_1)^2+(1-\epsilon_2)^2\big\}U_n^T R_n U_n \circ \mathcal{I}\big]R_n U_n. \quad (18)\end{aligned}$$

C. Proof

$(\partial L/\partial u_{kl}) = U_N \Gamma J^{mn} + J^{nm}\Gamma U_N^T$ is non-invertible.

By definition, $J^{mn} = \delta_{mk}\delta_{np}$. Let $F = \Gamma J^{mn}$. Then

$$F(x,y) = \begin{cases}\lambda_k, & \text{if } x = k \text{ and } y = p \\ 0, & \text{o.w.}\end{cases} \quad (19)$$

Therefore,

$$U_N \Gamma J^{mn}(x,y) = \begin{cases}\lambda_k u_{xk}, & \text{if } y = p \\ 0, & \text{o.w.}\end{cases} \quad (20)$$

So, $U_N\Gamma J^{mn}$ has one column (at $y = p$) that has a set of nonzero entries. It is evident that $H = (U_N\Gamma J^{mn})^T = J^{nm}\Gamma U_N^T$. Therefore, $(\partial L/\partial u_{kl})$ has one row and one column of possibly nonzero entries by construction. Assume m is that row index and the corresponding entry of H is $[H_{m1}, H_{m2}, \ldots, H_{mN}]$. Assume that $q \in \{1, 2, \ldots, N\}$ is the column index, where H receives entries from $U_N\Gamma J^{mn}$. One can find any two columns $i, j \in \{1, 2, \ldots, N\} - \{q\}$ such that the following transformation $C_i \longrightarrow (C_i/H_{mi})$ and $C_j \longrightarrow (C_j/H_{mj})$ would give two identical columns.

D. Proof of (14)

Let us assume A_0 is a real-valued, finite-dimensional ($\in \mathcal{R}^N$), positive semi-definite matrix with eigenvectors $U_0 = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N]$ and the corresponding eigenvalues $\lambda_0 = [\lambda_1, \lambda_2, \ldots, \lambda_N]$ (arranged in the decreasing order). In our case, $A_0 = L$, the graph Laplacian matrix. By definition $\lambda_N = 0$.

Let us also assume that A, U and λ are real-valued functions of a continuous parameter τ and they are analytic in the neighborhood of τ_0 such that $A(\tau_0) = A_0$, $U(\tau_0) = U_0$ and $\lambda(\tau_0) = \lambda_0$

$$\begin{aligned}A(\tau)\boldsymbol{u}_i(\tau) &= \lambda_i(\tau)\boldsymbol{u}_i(\tau) \\ \dot{A}(\tau)\boldsymbol{u}_i(\tau) + A(\tau)\dot{\boldsymbol{u}}_i &= \dot{\lambda}_i(\tau)\boldsymbol{u}_i + \lambda_i\dot{\boldsymbol{u}}_i. \quad (21)\end{aligned}$$

Now, because of the facts that $\boldsymbol{u}_i$ is analytic and $||\boldsymbol{u}_i|| = 1$ (orthonormal), $\dot{\boldsymbol{u}}_i \perp \boldsymbol{u}_i$. Taking inner product with $\boldsymbol{u}_i$ on both sides and plugging $\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_i\rangle = 0$, it can be found that

$$\begin{aligned}\big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_i\big\rangle + \big\langle A\dot{\boldsymbol{u}}_i, \boldsymbol{u}_i\big\rangle &= \big\langle \dot{\lambda}\boldsymbol{u}_i, \boldsymbol{u}_i\big\rangle \\ \big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_i\big\rangle &= \big\langle \dot{\lambda}\boldsymbol{u}_i, \boldsymbol{u}_i\big\rangle. \quad (22)\end{aligned}$$

Here, $\langle A\dot{\boldsymbol{u}}_i, \boldsymbol{u}_i\rangle = \langle \dot{\boldsymbol{u}}_i, A\boldsymbol{u}_i\rangle$ [because A is symmetric (self-adjoint)]. Using $A\boldsymbol{u}_i = \lambda_i\boldsymbol{u}_i$, we get $\langle \dot{\boldsymbol{u}}_i, A\boldsymbol{u}_i\rangle = \langle \dot{\boldsymbol{u}}_i, \lambda_i\boldsymbol{u}_i\rangle = \lambda_i\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_i\rangle = 0$. From (22), we obtain the expression for $\dot{\lambda}_i$ as $\dot{\lambda}_i = \langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_i\rangle$.

Taking the inner product of (21) with $\boldsymbol{u}_j$; $j \neq i$ and using the fact that $\boldsymbol{u}_i \perp \boldsymbol{u}_j$ we obtain

$$\begin{aligned}\big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_j\big\rangle + \big\langle A\dot{\boldsymbol{u}}_i, \boldsymbol{u}_j\big\rangle &= \lambda_i\big\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_j\big\rangle \\ \big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_j\big\rangle + \big\langle \dot{\boldsymbol{u}}_i, A\boldsymbol{u}_j\big\rangle &= \lambda_i\big\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_j\big\rangle \\ \big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_j\big\rangle + \big\langle \dot{\boldsymbol{u}}_i, \lambda_j\boldsymbol{u}_j\big\rangle &= \lambda_i\big\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_j\big\rangle \\ \big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_j\big\rangle &= \big(\lambda_i - \lambda_j\big)\big\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_j\big\rangle; \quad \lambda_i \neq \lambda_j \\ \big\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_j\big\rangle &= \frac{1}{\big(\lambda_i - \lambda_j\big)}\big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_j\big\rangle; \quad \lambda_i \neq \lambda_j. \quad (23)\end{aligned}$$

Equation (23) ($\lambda_i \neq \lambda_j$) and the fact that $\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_i\rangle = 0$ ($\lambda_i = \lambda_j$) suggest that $\dot{\boldsymbol{u}}_i$ can be expressed as a linear combination of eigenvectors $\boldsymbol{u}_j$; $j \in \{1, 2, \ldots, N\}$. Let us consider $\tau = a_{mn}$, where a_{mn} is an entry of A

$$\frac{\partial \boldsymbol{u}_i}{\partial a_{mn}} = \sum_{p\neq i}\frac{1}{\big(\lambda_i - \lambda_p\big)}\left\langle \frac{\partial A}{\partial a_{mn}}\boldsymbol{u}_i, \boldsymbol{u}_p\right\rangle \boldsymbol{u}_p; \quad \lambda_i \neq \lambda_p. \quad (24)$$

It can be seen that $\dot{\boldsymbol{u}}_i$ is spanned by the eigenvectors of A, when all the eigenvalues are numerically different. Due to the fact that A is real and symmetric, $(\partial A/\partial a_{mn})$ is also symmetric. Therefore, (24) can be rearranged as

$$\frac{\partial \boldsymbol{u}_i}{\partial a_{mn}} = \sum_{p\neq i}\frac{1}{\big(\lambda_i - \lambda_p\big)}\left\langle \boldsymbol{u}_i, \frac{\partial A}{\partial a_{mn}}\boldsymbol{u}_p\right\rangle \boldsymbol{u}_p; \quad \lambda_i \neq \lambda_p. \quad (25)$$

Next, we can use the fourth step of (23) to derive the expression of $\dot{\boldsymbol{u}}_i$ in case of $\lambda_i = \lambda_j$

$$\begin{aligned}&\big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_j\big\rangle = 0; \quad \lambda_i = \lambda_j \\ &\big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_j\big\rangle = \lambda_i\big\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_i\big\rangle; \quad \dot{\boldsymbol{u}}_i \perp \boldsymbol{u}_i \\ &\big\langle \boldsymbol{u}_i, \dot{A}\boldsymbol{u}_j\big\rangle = \lambda_i\big\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_i\big\rangle; \quad \dot{A} \text{ is symmetric} \\ &\big\langle \dot{A}\boldsymbol{u}_j, \boldsymbol{u}_i\big\rangle = \lambda_i\big\langle \dot{\boldsymbol{u}}_i, \boldsymbol{u}_i\big\rangle \\ &\dot{\boldsymbol{u}}_i = \begin{cases}\dfrac{1}{\lambda_i}\displaystyle\sum_{j\neq i}\big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_j\big\rangle\boldsymbol{u}_i; & \lambda_i = \lambda_j \neq 0 \\ \displaystyle\sum_{j\neq i}\big\langle \dot{A}\boldsymbol{u}_i, \boldsymbol{u}_j\big\rangle\boldsymbol{u}_i; & \lambda_i = \lambda_j = 0.\end{cases} \quad (26)\end{aligned}$$

Combining (25) and (26)

$$\begin{aligned}\frac{\partial \boldsymbol{u}_i}{\partial a_{mn}} &= \sum_{p\neq i}\frac{1-\delta\big(\lambda_i, \lambda_p\big)}{\big(\lambda_i - \lambda_p\big)}\left\langle \boldsymbol{u}_i, \frac{\partial A}{\partial a_{mn}}\boldsymbol{u}_p\right\rangle \boldsymbol{u}_p \\ &\quad + \sum_{\substack{q\neq i \\ \lambda_q \neq 0}}\frac{\delta\big(\lambda_i, \lambda_q\big)}{\lambda_i}\left\langle \boldsymbol{u}_i, \frac{\partial A}{\partial a_{mn}}\boldsymbol{u}_q\right\rangle \boldsymbol{u}_i \\ &\quad + \sum_{\substack{q\neq i \\ \lambda_q = 0}}\delta\big(\lambda_i, \lambda_q\big)\left\langle \boldsymbol{u}_i, \frac{\partial A}{\partial a_{mn}}\boldsymbol{u}_q\right\rangle \boldsymbol{u}_i. \quad (27)\end{aligned}$$

Here δ is the Kronecker delta function. By definition

$$\frac{\partial \boldsymbol{u}_k}{\partial w_j} = \sum_{m,n} \frac{\partial \boldsymbol{u}_k}{\partial L_{mn}} \frac{\partial L_{mn}}{\partial w_j} \tag{28}$$

where L_{mn} indicates the (m,n)th element of the symmetric matrix L. Let us first find the expression for $\lambda_i \neq \lambda_j$. Inserting (25) by replacing A with L to the above equation, we get

$$\begin{aligned}
\frac{\partial \boldsymbol{u}_k}{\partial w_j} &= \sum_{m,n} \left[\sum_{k \neq q} \frac{1}{(\lambda_k - \lambda_q)} \left\langle \frac{\partial L}{\partial L_{mn}} \boldsymbol{u}_k, \boldsymbol{u}_q \right\rangle \boldsymbol{u}_q \right] \frac{\partial L_{mn}}{\partial w_j} \\
&= \sum_{k \neq q} \left(\frac{1}{\lambda_k - \lambda_q} \right) \left[\sum_{m,n} \left\langle \frac{\partial L}{\partial L_{mn}} \boldsymbol{u}_k, \boldsymbol{u}_q \right\rangle \boldsymbol{u}_q \right] \frac{\partial L_{mn}}{\partial w_j} \\
&= \sum_{k \neq q} \left(\frac{1}{\lambda_k - \lambda_q} \right) \left[\sum_{m,n} \left\langle \boldsymbol{u}_k, \frac{\partial L}{\partial L_{mn}} \boldsymbol{u}_q \right\rangle \boldsymbol{u}_q \right] \frac{\partial L_{mn}}{\partial w_j} \\
&= \sum_{k \neq q} \left(\frac{1}{\lambda_k - \lambda_q} \right) \left\langle \boldsymbol{u}_k, \sum_{m,n} \left(\frac{\partial L}{\partial L_{mn}} \frac{\partial L_{mn}}{\partial w_j} \right) \boldsymbol{u}_q \right\rangle \boldsymbol{u}_q \\
&= \sum_{k \neq q} \left(\frac{1}{\lambda_k - \lambda_q} \right) \left\langle \boldsymbol{u}_k, B \frac{\partial W}{\partial w_j} B^T \boldsymbol{u}_q \right\rangle \boldsymbol{u}_q \\
&= \sum_{k \neq q} \left\langle \boldsymbol{u}_k, \frac{1}{(\lambda_k - \lambda_q)} B \frac{\partial W}{\partial w_j} B^T \boldsymbol{u}_q \right\rangle \boldsymbol{u}_q .
\end{aligned} \tag{29}$$

From (29), it can be observed that the eigenvector $\boldsymbol{u}_q$ is projected by the operator $(1/(\lambda_k - \lambda_q))B(\partial W/\partial w_j)B^T$ prior to the inner product. Applying the same definition the final expression can be found using (27)

$$\begin{aligned}
\frac{\partial \boldsymbol{u}_i}{\partial w_j} = & \sum_{\substack{p \neq i \\ \lambda_p \neq \lambda_i}} \frac{1 - \delta(\lambda_i, \lambda_p)}{(\lambda_i - \lambda_p)} \left\langle \boldsymbol{u}_i, B \frac{\partial W}{\partial w_j} B^T \boldsymbol{u}_p \right\rangle \boldsymbol{u}_p \\
& + \sum_{\substack{q \neq i \\ \lambda_q = \lambda_i \neq 0}} \frac{\delta(\lambda_i, \lambda_q)}{\lambda_i} \left\langle \boldsymbol{u}_i, B \frac{\partial W}{\partial w_j} B^T \boldsymbol{u}_q \right\rangle \boldsymbol{u}_i \\
& + \sum_{\substack{q \neq i \\ \lambda_q = \lambda_i = 0}} \delta(\lambda_i, \lambda_q) \left\langle \boldsymbol{u}_i, B \frac{\partial W}{\partial w_j} B^T \boldsymbol{u}_q \right\rangle \boldsymbol{u}_i .
\end{aligned} \tag{30}$$

Code Availability

The code is made publicly available in Code Ocean (https://codeocean.com/capsule/7564483/tree) and github (https://github.com/50-Cent/PrecoG).

References

[1] B. Widrow and M. E. J. Hoff, "Adaptive switching circuits," in *Proc. IRE WESCON Conv. Rec., Western Electron. Show Conv.* Los Angeles, CA, USA: Institute of Radio Engineers, Aug. 1960, pp. 1–9.

[2] B. Farhang-Boroujeny, "Fast LMS/Newton algorithms based on autoregressive modeling and their application to acoustic echo cancellation," *IEEE Trans. Signal Process.*, vol. 45, no. 8, pp. 1987–2000, Aug. 1997.

[3] H. Deng and M. Doroslovacki, "Proportionate adaptive algorithms for network echo cancellation," *IEEE Trans. Signal Process.*, vol. 54, no. 5, pp. 1794–1803, May 2006.

[4] F. Albu and H. K. Kwan, "Combined echo and noise cancellation based on Gauss–Seidel pseudo affine projection algorithm," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 3, May 2004, p. III-505.

[5] C. Paleologu, S. Ciochină, and J. Benesty, "An efficient proportionate affine projection algorithm for echo cancellation," *IEEE Signal Process. Lett.*, vol. 17, no. 2, pp. 165–168, Feb. 2010.

[6] B. Widrow, J. McCool, and B. Medoff, "Adaptive control by inverse modeling," in *Proc. 12th Asilomar Conf. Circuits, Syst., Comput.*, 1978, p. 90.

[7] W. Chen, T. Nemoto, T. Kobayashi, T. Saito, E. Kasuya, and Y. Honda, "ECG and heart rate detection of prenatal cattle foetus using adaptive digital filtering," in *Proc. 22nd Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, vol. 2, Jul. 2000, pp. 962–965.

[8] P. He, G. Wilson, and C. Russell, "Removal of ocular artifacts from electro-encephalogram by adaptive filtering," *Med. Biol. Eng. Comput.*, vol. 42, no. 3, pp. 407–412, 2004.

[9] C. Paleologu, S. Ciochina, A. A. Enescu, and C. Vladeanu, "Gradient adaptive lattice algorithm suitable for fixed point implementation," in *Proc. 3rd Int. Conf. Digit. Telecommun. (ICDT)*, Jun. 2008, pp. 41–46.

[10] H. Fan and X. Q. Liu, "GAL and LSL revisited: New convergence results," *IEEE Trans. Signal Process.*, vol. 41, no. 1, pp. 55–66, Jan. 1993.

[11] S. Haykin, *Adaptive Filter Theory*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[12] Y. V. Zakharov, G. P. White, and J. Liu, "Low-complexity RLS algorithms using dichotomous coordinate descent iterations," *IEEE Trans. Signal Process.*, vol. 56, no. 7, pp. 3150–3161, Jul. 2008.

[13] F. Albu and H. K. Kwan, "Fast block exact Gauss–Seidel pseudo affine projection algorithm," *Electron. Lett.*, vol. 40, no. 22, pp. 1451–1453, Oct. 2004.

[14] A. Gonzalez, M. Ferrer, F. Albu, and M. De Diego, "Affine projection algorithms: Evolution to smart and fast algorithms and applications," in *Proc. 20th Eur. Signal Process. Conf. (EUSIPCO)*, 2012, pp. 1965–1969.

[15] S. L. Gay, "The fast affine projection algorithm," in *Acoustic Signal Processing for Telecommunication*. Boston, MA, USA: Springer, 2000, pp. 23–45.

[16] S. Zhao, Z. Man, S. Khoo, and H. R. Wu, "Stability and convergence analysis of transform-domain LMS adaptive filters with second-order autoregressive process," *IEEE Trans. Signal Process.*, vol. 57, no. 1, pp. 119–130, Jan. 2009.

[17] M. H. Costa, J. C. M. Bermudez, and N. J. Bershad, "Stochastic analysis of the LMS algorithm with a saturation nonlinearity following the adaptive filter output," *IEEE Trans. Signal Process.*, vol. 49, no. 7, pp. 1370–1387, Jul. 2001.

[18] D. I. Kim and P. D. Wilde, "Performance analysis of the DCT-LMS adaptive filtering algorithm," *Signal Process.*, vol. 80, no. 8, pp. 1629–1654, Aug. 2000.

[19] C. R. C. Nakagawa, S. Nordholm, F. Albu, and W.-Y. Yan, "Closed-loop feedback cancellation utilizing two microphones and transform domain processing," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2014, pp. 3645–3649.

[20] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*. Hoboken, NJ, USA: Wiley, 2013, ch. 7.

[21] F. Beaufays, "Transform-domain adaptive filters: An analytical approach," *IEEE Trans. Signal Process.*, vol. 43, no. 2, pp. 422–431, Feb. 1995.

[22] K. Chen, *Matrix Preconditioning Techniques and Applications*, vol. 19. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[23] M. Benzi, "Preconditioning techniques for large linear systems: A survey," *J. Comput. Phys.*, vol. 182, no. 2, pp. 418–477, 2002.

[24] Y. Cao, M.-Q. Jiang, and Y.-L. Zheng, "A splitting preconditioner for saddle point problems," *Numer. Linear Algebra With Appl.*, vol. 18, no. 5, pp. 875–895, Oct. 2011.

[25] Y. Dauphin, H. de Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1504–1512.

[26] O. Chapelle and D. Erhan, "Improved preconditioner for hessian free optimization," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, vol. 201, no. 1, pp. 1–8.

[27] A. Dziekonski, A. Lamecki, and M. Mrozowski, "Jacobi and Gauss–Seidel preconditioned complex conjugate gradient method with GPU acceleration for finite element method," in *Proc. 40th Eur. Microw. Conf. (EuMC)*, Sep. 2010, pp. 1305–1308.

[28] W.-P. Tang, "Toward an effective sparse approximate inverse preconditioner," *SIAM J. Matrix Anal. Appl.*, vol. 20, no. 4, pp. 970–986, Jan. 1999.

[29] M. Benzi, J. K. Cullum, and M. Tuma, "Robust approximate inverse preconditioning for the conjugate gradient method," *SIAM J. Sci. Comput.*, vol. 22, no. 4, pp. 1318–1332, Jan. 2000.

[30] E. Chow and A. Patel, "Fine-grained parallel incomplete LU factorization," *SIAM J. Sci. Comput.*, vol. 37, no. 2, pp. C169–C193, Jan. 2015.
[31] S. Yang, L. Li, S. Wang, W. Zhang, and Q. Huang, "A graph regularized deep neural network for unsupervised image representation learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1203–1211.
[32] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.
[33] H. P. Maretic, D. Thanou, and P. Frossard, "Graph learning under sparsity priors," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 6523–6527.
[34] E. Pavez, H. E. Egilmez, and A. Ortega, "Learning graphs with monotone topology properties and multiple connected components," *IEEE Trans. Signal Process.*, vol. 66, no. 9, pp. 2399–2413, May 2018.
[35] M. G. Rabbat, "Inferring sparse graphs from smooth signals with theoretical guarantees," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 6533–6537.
[36] J. Abernethy, O. Chapelle, and C. Castillo, "Graph regularization methods for web spam detection," *Mach. Learn.*, vol. 81, no. 2, pp. 207–225, Nov. 2010.
[37] D. Hallac, J. Leskovec, and S. Boyd, "Network lasso: Clustering and optimization in large graphs," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2015, pp. 387–396.
[38] C. J. Stam, "Modern network science of neurological disorders," *Nature Rev. Neurosci.*, vol. 15, no. 10, pp. 683–695, 2014.
[39] C. I. Bargmann and E. Marder, "From the connectome to brain function," *Nature Methods*, vol. 10, no. 6, p. 483, 2013.
[40] M. Rubinov and O. Sporns, "Complex network measures of brain connectivity: Uses and interpretations," *NeuroImage*, vol. 52, no. 3, pp. 1059–1069, 2010.
[41] B. Widrow, Y. Kim, D. Park, and J. K. Perin, "Nature's learning rule: The Hebbian-LMS algorithm," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Amsterdam, The Netherlands: Elsevier, 2019, pp. 1–30.
[42] T. Batabyal, R. Sarkar, and S. T. Acton, "GraDED: A graph-based parametric dictionary learning algorithm for event detection," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 1797–1801.
[43] M.-D. Choi, "Tricks or treats with the Hilbert matrix," *Amer. Math. Monthly*, vol. 90, no. 5, pp. 301–312, May 1983.
[44] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. Hoboken, NJ, USA: Wiley, 1949.
[45] G. G. Turrigiano and S. B. Nelson, "Homeostatic plasticity in the developing nervous system," *Nature Rev. Neurosci.*, vol. 5, no. 2, pp. 97–107, 2004.
[46] C. C. Swanwick, N. R. Murthy, and J. Kapur, "Activity-dependent scaling of GABAergic synapse strength is regulated by brain-derived neurotrophic factor," *Mol. Cellular Neurosci.*, vol. 31, no. 3, pp. 481–492, Mar. 2006.
[47] E. Oja, "Simplified neuron model as a principal component analyzer," *J. Math. Biol.*, vol. 15, no. 3, pp. 267–273, 1982.

Tamal Batabyal received the B.Tech. degree in electrical engineering from the Indian School of Mines (IIT), Dhanbad, India, in 2010, the M.Tech. degree in computer science from the Indian Statistical Institute, Kolkata, India, in 2014, and the Ph.D. degree in electrical engineering from the University of Virginia, Charlottesville, VA, USA, in 2019.

He is currently a Post-Doctoral Research Associate in neurology with the University of Virginia. He is working under the supervision of Dr. Jaideep Kapur. His current research interests include graph theoretic analysis and modeling, image processing using deep learning, parallels between neuronal and neural networks, and experimental neuroscientific problems in seizure and epilepsy.

Daniel Weller (Senior Member, IEEE) received the B.S. degree (Hons.) in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in May 2006, and the S.M. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in June 2008 and June 2012, respectively.

From 2012 to 2014, he was a Post-Doctoral Fellow, supported by the National Institutes of Health (NIH) National Research Service Award, with the University of Michigan, Ann Arbor, MI, USA, where he worked with Jeffrey Fessler and Douglas Noll. From 2014 to 2020, he was an Assistant Professor of electrical and computer engineering with the University of Virginia, Charlottesville, VA, USA. His lab focused on image reconstruction, processing, and analysis, such as fast and motion-robust magnetic resonance imaging for cardiac and body-imaging applications, and enhancement and visualization of optical microscope images for neuroscience. He worked at the Massachusetts Institute of Technology with Vivek Goyal (now at Boston University, Boston, MA, USA). He currently works with the AI Modeling and Center of Excellence, KLA Corporation, Ann Arbor. KLA is a global leader in diversified electronics for the semiconductor manufacturing ecosystem. As a Senior AI/Algorithms Engineering Manager, he leads teams of talented engineers developing algorithms for current and future products.

Dr. Weller was inducted into the Tau Beta Pi and the Eta Kappa Nu.

Jaideep Kapur received the M.B.B.S. degree from the University of Delhi, New Delhi, India, in 1984, and Ph.D. degree from the University of Virginia, Charlottesville, VA, USA, in 1988.

He is currently the Eugene Meyer III Professor of neuroscience, a Professor of neurology, and the Brain Institute Director with the University of Virginia. He is also a Clinician Investigator, whose laboratory studies the neurobiological mechanisms underlying status epilepticus and neuronal circuits active during seizures. He was a Leader of Established Status Epilepticus Treatment Trial (ESETT), the National Institutes of Health (NIH)-supported 60 site, randomized, blinded, and comparative effectiveness study of three drugs for status epilepticus.

Dr. Kapur received the 2013 Epilepsy Research Recognition Award for Basic Science conferred by the American Epilepsy Society, the Ambassador For Epilepsy Award from the International League Against Epilepsy in 2017, and the Jacob Javits Award from the National Institute of Neurological Disorders and Stroke in 2021. He served as the President for the American Epilepsy Society in 2010 and on the Editorial Board of journals, such as *Neurology*, *Annals of Neurology*, *Experimental Neurology*, *Epilepsy Research*, and *Epilepsy Currents*, and multiple NIH review panels.

Scott T. Acton (Fellow, IEEE) received the B.S. degree from Virginia Tech, Blacksburg, VA, USA, in 1988, and the M.S. and Ph.D. degrees from The University of Texas at Austin, Austin, TX, USA, in 1990 and 1993, respectively.

He is currently the Program Director in computer and information science and engineering with the U.S. National Science Foundation. He is also a Professor of electrical and computer engineering and biomedical engineering with the University of Virginia (UVA), Charlottesville, VA, USA. His laboratory at UVA is called Virginia Image and Video Analysis (VIVA). VIVA specializes in bioimage analysis problems. The research emphases of VIVA include image analysis in bioimaging, neuroscience, video analysis in education, tracking, segmentation, representation, retrieval, classification, and enhancement. Recent theoretical interests include active contours, level sets, partial differential equation methods, scale space methods, graph signal processing, and machine learning. He has over 300 publications in the image analysis area, including the books *Biomedical Image Analysis: Tracking* and *Biomedical Image Analysis: Segmentation*. His other interests include mountain biking and hiking in the Blue Ridge Mountains.

Prof. Acton is a fellow of IEEE for contributions to biomedical image analysis. He was the 2018 Co-Chair of the IEEE International Symposium on Biomedical Imaging. He was recently the Editor-in-Chief of the IEEE TRANSACTIONS ON IMAGE PROCESSING from 2014 to 2018.