# Graph Unrolling Networks: Interpretable Neural Networks for Graph Signal Denoising

Siheng Chen, *Member, IEEE*, Yonina C. Eldar, *Fellow, IEEE*, and Lingxiao Zhao

***Abstract*—We propose an interpretable graph neural network framework to denoise single or multiple noisy graph signals. The proposed *graph unrolling networks* expand algorithm unrolling to the graph domain and provide an interpretation of the architecture design from a signal processing perspective. We unroll an iterative denoising algorithm by mapping each iteration into a single network layer where the feed-forward process is equivalent to iteratively denoising graph signals. We train the graph unrolling networks through unsupervised learning, where the input noisy graph signals are used to supervise the networks. By leveraging the learning ability of neural networks, we adaptively capture appropriate priors from input noisy graph signals. A core component of graph unrolling networks is the *edge-weight-sharing graph convolution operation*, which parameterizes each edge weight by a trainable kernel function whose trainable parameters are shared by all the edges. The proposed convolution is permutation-equivariant and can flexibly adjust the edge weights to various graph signals. We further consider two special cases of this class of networks, graph unrolling sparse coding (GUSC) and graph unrolling trend filtering (GUTF), by unrolling sparse coding and trend filtering, respectively. To validate the proposed methods, we conduct extensive experiments on both real-world datasets and simulated datasets, and demonstrate that our methods have smaller denoising errors than conventional denoising algorithms and state-of-the-art graph neural networks. For denoising a single smooth graph signal, the normalized mean square error of the proposed networks is around 40% and 60% lower than that of graph Laplacian denoising and graph wavelets, respectively.**

***Index Terms*—Graph neural networks, algorithm unrolling, graph signal denoising, graph convolution, weight sharing.**

## I. Introduction

DATA today is often generated from diverse sources, including social, citation, biological networks and physical infrastructure [1]. Unlike time-series signals or images, such signals possess complex and irregular structures, which can be modeled as graphs. Analyzing graph signals requires dealing with the underlying irregular relationships. Graph signal processing generalizes the classical signal processing toolbox to the graph domain and provides a series of techniques to process graph signals [1], including graph-based transformations [2], [3], sampling and recovery of graph signals [4], [5] and graph topology learning [6]. Graph neural networks provide a powerful framework to learn from graph signals with graphs as induced biases [7]. Permeating the benefits of deep learning to the graph domain, graph convolutional networks and variants have attained remarkable success in social network analysis [8], point cloud processing [9] and computer vision [10].

In this work, we consider denoising graph signals [11], [12]. In classical signal processing, signal denoising is one of the most ubiquitous tasks [13]. To handle graph signals, there are two mainstream approaches: graph-regularization-based optimization and graph dictionary design. The optimization approach usually introduces a graph-regularization term that promotes certain properties of the graph signal and solves a regularized optimization problem to obtain a denoised solution [12], [14], [15]. In image denoising, the total variation, which captures the integral of the absolute gradient of the image [16], [17], is often used. Minimizing the total variation of an image helps remove unwanted noise while preserving important details, such as edges and contours. In graph signal denoising, the regularizers are often chosen to relate to the graph properties. For example, a popular choice is a quadratic form of the graph Laplacian, which captures the second-order difference of a graph signal, corresponding to a graph smoothness prior [11], [18]. Graph total variation captures the sparsity of the first-order difference of a graph signal, reflecting piecewise-constant prior [14], [15], [19].

In comparison, the graph-dictionary approach aims to reconstruct graph signals through a predesigned graph dictionary, such as graph wavelets [2], windowed graph Fourier transforms [20], and graph frames [21]. These dictionaries are essentially variants of graph filters. A reconstructed graph signal consists of a sparse combination of elementary graph signals in a graph dictionary. The combination coefficients can be obtained through sparse coding algorithms, such as matching pursuit and basis pursuit [22], [23]. A fast implementation of graph dictionary are graph filter banks, which use a series of band-pass graph filters that expands the input graph signal into multiple subband components [3], [21], [24], [25]. By adjusting the component in each subband, a graph filter bank can flexibly modify a

Manuscript received June 2, 2020; revised March 3, 2021 and May 13, 2021; accepted May 25, 2021. Date of publication June 11, 2021; date of current version July 2, 2021. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Jarvis Haupt. This work was supported by Mitsubishi Electric Research Laboratories (MERL). This work was mainly done while Siheng Chen was working at MERL. *(Corresponding author: Siheng Chen.)*

Siheng Chen is with the Shanghai Jiao Tong University and Shanghai Artificial Intelligence Laboratory, Shanghai 200240, China (e-mail: sihengc@sjtu.edu.cn).

Yonina C. Eldar is with the Department of Math and CS, Weizmann Institute of Science, Rehovot 7610001, Israel (e-mail: yonina.eldar@weizmann.ac.il).

Lingxiao Zhao is with the Heinz College, Carnegie Mellon University, Pittrsbugh, PA 15213 USA (e-mail: lingxiao@cmu.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TSP.2021.3087905, provided by the authors.

Digital Object Identifier 10.1109/TSP.2021.3087905

graph signal and suppress noise, especially in the high-frequency band.

A fundamental challenge for both denoising approaches is that we may not know an appropriate prior on the noiseless graph signals in practice. It is then hard to either choose an appropriate graph-regularization term or design an appropriate graph dictionary. Furthermore, some graph priors are too complicated to be precisely described in mathematical terms or may lead to computationally intensive algorithms.

To solve this issue, it is desirable to learn an appropriate prior from given graph signals; in other words, the denoising algorithm should have sufficient flexibility to learn from and adapt to arbitrary signal priors. Deep neural networks have demonstrated strong power in learning abstract, yet effective features from a huge amount of data [26]. As the extension of neural networks to the graph domain, graph neural networks have also received a lot of attention and achieved significant success in social network analysis and geometric data analysis [8], [27]. One mainstream graph neural network architecture is the graph convolutional network (GCN), which relies on a layered architecture that consists of trainable graph convolution operations, followed by pointwise nonlinear functions [8], [28], [29]. Some variants include graph attention networks [30], deep graph infomax [31], simple graph convolution [32], and the graph U-net [33]. These models have shown remarkable success in graph-based semi-supervised learning [8], [30] and graph classification tasks [34]. However, these neural network architectures are typically designed through trial and error. It is thus hard to explain the design rationale and further improve the architectures [35], [36].

In this work, we leverage the powerful learning ability of graph neural networks and combine them with interpretablity based on a signal processing perspective. Furthermore, most graph neural networks are developed for supervised-learning tasks, such node classification [8], link prediction [37] and graph classification [33]. Those tasks require a large number of ground-truth labels, which is expensive to obtain. Here we consider an unsupervised-learning setting, where the networks have to learn from a few noisy graph signals and the ground-truth noiseless graph signals are unknown. Through unsupervised learning, we demonstrate the generalization ability of the proposed graph neural networks.

Our goal is to develop a framework for graph network denoising by combining the advantages of both conventional graph signal denoising algorithms and graph neural networks. On the one hand, we follow the iterative procedures of conventional denoising algorithms, which provides interpretability and explicit graph regularization; on the other hand, we parameterize a few mathematically-designed operations through neural networks and train the parameters, which provides flexibility and learning ability. We bridge between conventional graph signal denoising algorithms and graph neural networks by using the powerful framework of algorithm unrolling [38]. It provides a concrete and systematic connection between iterative algorithms in signal processing and deep neural networks, and paves the way to developing interpretable network architectures. Related unrolling techniques have been successfully applied in many problem areas, such as sparse coding [39], ultrasound signal processing [35], power system [40] image deblurring [41], [42], image denoising [43], and super resolution [44].

In this work, we expand algorithm unrolling to the graph domain. We first propose a general iterative algorithm for graph signal denoising and then transform it to a graph neural network through algorithm unrolling, where each iteration is mapped to a network layer. Compared to conventional denoising algorithms [12], the proposed graph unrolling network is able to learn a variety of priors from given graph signals by leveraging deep neural networks. Compared to many other graph neural networks [8], the proposed graph unrolling network is interpretable by following analytical iterative steps. To train graph unrolling networks, we use single or multiple noisy graph signals and minimize the difference between the original input, which is the noisy graph signal, and the network output, which is the denoised graph signal; in other words, the input noisy measurements are used to supervise the neural network training. Surprisingly, even when we train until convergence, the output does not overfit the noisy input in most cases. The intuition is that the proposed operations and architectures carry implicit graph regularization and thus avoid overfitting.

A core component in the proposed graph unrolling networks is the edge-weight-sharing graph convolution operation. The proposed graph convolution parameterizes edge weights by a trainable kernel function, which maps a pair of vertex coordinates in the graph spectral domain to an edge weight. Since the trainable parameters in the kernel function are shared by all the edges, this graph convolution has a weight-sharing property. We also show that it is equivariant to the permutation of vertices. Our convolution is different from conventional graph filtering [12], as it includes trainable parameters that can transform a graph signal from the original graph vertex domain to a feature domain. It is also different from many trainable graph convolutions [8], since it adjusts edge weights according to the input graph signals during training, which makes it flexible to capture complicated signal priors.

Based on the graph unrolling network framework, we further propose two specific architectures by unrolling graph sparse coding and graph trend filtering. These two are typical denoising algorithms based on graph dictionary design and graph-regularization-based optimization, respectively. We consider both networks to demonstrate the generalization of the proposed framework.

To validate the performance of the proposed methods, we conduct a series of experiments on both simulated and real-world datasets with Gaussian noise, mixture noise and Bernoulli noise. We show that graph unrolling networks consistently achieve better denoising performance than conventional graph signal denoising algorithms and state-of-the-art graph neural networks on various types of graph signals and noise models. Even for denoising a single smooth graph signal, the proposed graph unrolling networks are around 40% and 60% better than graph Laplacian denoising [11] and graph wavelets [2], respectively. This demonstrates that the unrolling approach allows to obtain improved results over existing methods even using a single training point.

The main contributions of this work include:

- We propose interpretable graph unrolling networks by unrolling a general iterative algorithm for graph signal denoising in an unsupervised-learning setting;
- We propose a trainable edge-weight-sharing graph convolution whose trainable parameters are shared across all the edges. It is equivariant to permutation of vertices;
- We propose two specific network architectures under the umbrella of graph unrolling networks: graph unrolling sparse coding and graph unrolling trend filtering; and
- We conduct experiments on both simulated and real-world data to validate that the proposed denoising methods outperform both conventional graph signal denoising methods and state-of-the-art graph neural networks on various types of graph signals and noise models.

The rest of the paper is organized as follows: Section II formulates the graph signal denoising problem and revisits a few classical denoising methods. Section III proposes an edge-weight-sharing graph convolution operation, which is a core operation in the proposed network. Section IV describes the general framework of graph unrolling networks and provides two specific architectures: graph sparse coding and graph trend filtering. Experiments validating the advantages of our methods are provided in Section V.

## II. Problem Formulation

In this section, we mathematically formulate the task of graph signal denoising and review a few classical denoising methods, which lay the foundation for the proposed methods.

We consider a graph $G = (\mathcal{V}, \mathcal{E}, \mathrm{A})$, where $\mathcal{V} = \{v_n\}_{n=1}^N$ is the set of *vertices*, $\mathcal{E} = \{e_m\}_{m=1}^M$ is the set of undirected *edges*, and $\mathrm{A} \in \mathbb{R}^{N \times N}$ is the graph adjacency matrix, representing connections between vertices. The weight $\mathrm{A}_{i,j}$ of an edge from the $i$th to the $j$th vertex characterizes the relation, such as similarity or dependency, between the corresponding signal values. Using the graph representation $G$, a *graph signal* is defined as a map that assigns a signal coefficient $x_n \in \mathbb{R}$ to the vertex $v_n$. A graph signal can be written as a length-$N$ vector defined by $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_N \end{bmatrix}^T$, where the $n$th vector element $x_n$ is indexed by the vertex $v_n$.

Multiplication between the graph adjacency matrix and a graph signal, $\mathrm{A}\,\mathbf{x}$, replaces the signal coefficient at each vertex with the weighted linear combination of the signal coefficients at the corresponding neighbors. In other words, the graph adjacency matrix enables the value at each vertex shift to its neighbors; we thus call it a *graph shift operator* [45]. Some other choices of a graph shift operator could be the graph Laplacian or the graph transition matrix [1]. In order for the output norm not to increase after graph shifting, we normalize the graph adjacency matrix, $\mathrm{A}^{\mathrm{norm}} = \mathrm{A}\,/|\lambda_{\max}(\mathrm{A})|$, where $\lambda_{\max}(\mathrm{A})$ denotes the eigenvalue of A with the largest magnitude. The normalized matrix has spectral norm $\|\,\mathrm{A}^{\mathrm{norm}}\,\|_2 = 1$. In this paper, we assume that the all graph shift operators are normalized, that is $\mathrm{A} = \mathrm{A}^{\mathrm{norm}}$.

Assume that we are given a length-$N$ noisy measurement

$$\mathbf{t} = \mathbf{x} + \mathbf{e}, \tag{1}$$

where $\mathbf{x}$ is the noiseless graph signal and $\mathbf{e}$ is noise. The goal of graph signal denoising is to recover $\mathbf{x}$ from $\mathbf{t}$ by removing the noise. We can further extend this setting to multiple graph signals. Consider $K$ measurements in a $N \times K$ matrix, $\mathrm{T} = \begin{bmatrix} \mathbf{t}^{(1)} & \mathbf{t}^{(2)} & \dots & \mathbf{t}^{(K)} \end{bmatrix} = \mathrm{X} + \mathrm{E}$, where $\mathrm{X} = \begin{bmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \dots & \mathbf{x}^{(k)} & \dots & \mathbf{x}^{(K)} \end{bmatrix}$ is a matrix of $K$ noiseless graph signals, and E is a $N \times K$ matrix that contains independent and identically distributed random noise. We assume that all graph signals in X are generated on the same graph and share related graph-based properties. We thus aim to recover X from T by removing the noise E.

Without any prior information on the noiseless graph signals, it is impossible to split noise from the measurements. Possible priors include sparsity, graph smoothness and graph piecewise-smoothness [46]. Here we consider a general graph signal model in which the graph signal is generated through graph filtering over vertices; that is, $\mathbf{x} = \mathbf{h} *_v \mathbf{s} = \sum_{\ell=1}^L h_\ell\, \mathrm{A}^\ell\, \mathbf{s}$, where $\mathbf{s} \in \mathbb{R}^N$ is a base graph signal, which may not have any graph-related properties, $*_v$ indicates a convolution on the graph vertex domain and $\mathbf{h} = \begin{bmatrix} h_1 & h_2 & \cdots & h_L \end{bmatrix}^T \in \mathbb{R}^L$ are the predesigned and fixed filter coefficients with $L$ the filter length. Here we consider a typical design of a graph filter, which is a polynomial of the graph shift; see detailed discussion in Section III. The graph filtering process modifies a given base graph signal according to certain patterns of the graph and explicitly regularizes the output.

Based on this graph signal model, we can remove noise by solving the following optimization problem:

$$\min_{\mathbf{s} \in \mathbb{R}^N} \frac{1}{2} ||\mathbf{t} - \mathbf{x}||_2^2 + u(\mathrm{P}\,\mathbf{x}) + r(\mathrm{Q}\,\mathbf{s}), \quad \text{subject to } \mathbf{x} = \mathbf{h} *_v \mathbf{s}, \tag{2}$$

where $u(\cdot), r(\cdot) \in \mathbb{R}$ are additional regularization terms on $\mathbf{s}$ and $\mathbf{x}$ respectively and P and Q are two matrices. The denoised graph signal is then given by $\mathbf{h} *_v \mathbf{s}$.

To connect the general graph signal denoising problem (2) to previous works, we present several special cases of (2).

*1) Graph Sparse Coding:* Here we consider reconstructing a noiseless graph signal through graph filtering and regularizing the graph signal to be sparse. The optimization problem of graph sparse coding is

$$\min_{\mathbf{s} \in \mathbb{R}^N} \frac{1}{2} ||\mathbf{t} - \sum_{\ell=1}^L h_\ell\, \mathrm{A}^\ell\, \mathbf{s}||_2^2 + \alpha\, \|\mathbf{s}\|_1\,. \tag{3}$$

In this setting, $\mathbf{x} = \mathbf{h} *_v \mathbf{s} = \sum_{\ell=1}^L h_\ell\, \mathrm{A}^\ell\, \mathbf{s}$, $u(\cdot) = 0, r(\cdot) = \alpha \|\cdot\|_1$ and $\mathrm{P} = \mathrm{Q} = \mathrm{I}$. The term $\|\mathbf{s}\|_1$ promotes sparsity of the base signal and $\mathbf{h} *_v \mathbf{s}$ allows the sparse signal coefficients to diffuse over the graph. Many denoising algorithms based on graph filter banks and graph dictionary representations are variations of graph sparse coding [46]. They design various graph filters to adjust subbbands' responses and use matching pursuit or basis pursuit to solve (3).

*2) Graph Laplacian Denoising:* Here we consider using the the second-order difference to regularize a graph signal. The optimization problem of graph Laplacian denoising is

$$\min_{\mathbf{x} \in \mathbb{R}^N} \frac{1}{2} ||\mathbf{t} - \mathbf{x}||_2^2 + \alpha \mathbf{x}^T \mathcal{L} \mathbf{x}, \tag{4}$$

where $\mathcal{L} = \mathrm{D} - \mathrm{A} \in \mathbb{R}^{N \times N}$ is the graph Laplacian matrix with diagonal degree matrix $\mathrm{D}_{i,i} = \sum_j \mathrm{A}_{i,j}$. In this setting, $\mathbf{x} = \mathbf{h} *_v$

$\mathbf{s}=\mathbf{s}, u(\cdot)=\alpha\|\cdot\|_2^2, r(\mathbf{s})=0, \mathrm{P}=\mathcal{L}^{\frac{1}{2}}$ and $\mathrm{Q}=\mathrm{I}$. Here we do not consider the effect of graph filtering and directly regularize the graph signal $\mathbf{x}$. The term

$$u(\mathrm{P}\,\mathbf{x})=\alpha\left\|\mathcal{L}^{\frac{1}{2}}\mathbf{x}\right\|_2^2=\alpha\mathbf{x}^T\mathcal{L}\mathbf{x}=\alpha\sum_{(i,j)\in\mathcal{E}}\mathrm{A}_{i,j}(x_i-x_j)^2,$$

is well known as the quadratic form of the graph Laplacian, which has been widely used in graph-based semi-supervised learning, spectral clustering and graph signal processing [1], [47], [48]. It captures the second-order difference of a graph signal by accumulating the pairwise differences between signal values associated with adjacent vertices. When solving (4), we regularize $\mathbf{x}$ to be smooth [47].

*3) Graph Trend Filtering:* Here we consider using the first-order difference to regularize a graph signal. The optimization problem is

$$\min_{\mathbf{x}\in\mathbb{R}^N}\frac{1}{2}||\mathbf{t}-\mathbf{x}||_2^2+\alpha\left\|\Delta\mathbf{x}\right\|_1, \tag{5}$$

where $\Delta$ is a $M\times N$ graph incidence matrix with $M$ the number of edges and $N$ the number of nodes. Each row of $\Delta$ corresponds to an edge. For example, if $e_i$ is an edge that connects the $j$th vertex to the $k$th vertex ($j<k$), the elements of the $i$th row of $\Delta$ are

$$\Delta_{i,\ell}=\begin{cases}-\sqrt{\mathrm{A}_{j,k}}, & \ell=j;\\ \sqrt{\mathrm{A}_{j,k}}, & \ell=k;\\ 0, & \text{otherwise.}\end{cases} \tag{6}$$

The graph incident matrix measures the first-order difference and satisfies $\Delta^T\Delta=\mathcal{L}$. In this setting $\mathbf{x}=\mathbf{h}*_v\mathbf{s}=\mathbf{s}$, $u(\cdot)=\alpha\|\cdot\|_1$, $r(\cdot)=0$, $\mathrm{P}=\Delta$ and $\mathrm{Q}=\mathrm{I}$. The term

$$u(\mathrm{P}\,\mathbf{x})=\alpha\left\|\Delta\mathbf{x}\right\|_1=\alpha\sum_{(i,j)\in\mathcal{E}}\mathrm{A}_{i,j}\left|x_i-x_j\right|,$$

is known as the graph total variation and is often used in graph signal denoising. Similar to the graph-Laplacian regularization, the graph total variation considers pairwise differences. However, it uses the $\ell_1$ norm to promote sparsity of the first-order differences. When solving (5), we encourage $\mathbf{x}$ to be piecewise-constant [14], [15].

Section IV solves the general denoising problem (2) through algorithm unrolling and proposes a framework for developing graph unrolling networks. Before that, we first propose a core component of graph unrolling networks: graph convolution.

## III. Edge-Weight-Sharing Graph Convolution

In this section, we present a graph convolution operation, which can be trained in end-to-end learning. The proposed convolution parameterizes each edge weight by a kernel function whose trainable parameters are shared across all the edges; we thus call it *edge-weight-sharing graph convolution*. It will be used as a building block of the graph unrolling networks in Section IV. Here we first revisit the standard graph convolution used in signal processing, and then equip it with trainable parameters, suitable for neural networks.

### A. Graph Convolution in Signal Processing

We first revisit 1D cyclic convolution in conventional discrete signal processing. Let $\mathbf{x}\in\mathbb{R}^N$ be a time-series. The output time-series after cyclic convolution is $\mathbf{y}=\mathbf{h}*\mathbf{x}=\sum_{\ell=1}^{L}h_\ell\,\mathrm{C}^\ell\,\mathbf{x}\in\mathbb{R}^N$, where $L$ is the length of a filter, $\mathbf{h}=\begin{bmatrix}h_1 & h_2 & \cdots h_L\end{bmatrix}^T\in\mathbb{R}^L$ are the filter coefficients and the cyclic-permutation matrix

$$\mathrm{C}=\begin{bmatrix}0 & 0 & \cdots & 0 & 1\\ 1 & 0 & \cdots & 0 & 0\\ 0 & 1 & \ddots & 0 & 0\\ \vdots & \vdots & \ddots & \ddots & \\ 0 & 0 & \ddots & 1 & 0\end{bmatrix}\in\mathbb{R}^{N\times N}$$

is a matrix representation of a directed cyclic graph [45]. It reflects the underlying structure of a finite, periodic discrete time series. All edges are directed and have the same weight 1, reflecting the causality of a time series. A polynomial of the cyclic-permutation matrix, $\sum_{\ell=1}^{L}h_\ell\,\mathrm{C}^\ell$, is a filter in the time domain. The essence of convolution/filtering is to update a signal coefficient by weighted averaging of the neighboring coefficients. The neighbors are defined based on the cyclic-permutation matrix C where the weights, or the filter coefficients, are shared across the entire signal.

We can use a mathematical analogy to generalize convolution from the time domain to the graph domain; that is, we simply replace the cyclic-permutation matrix C by the graph adjacency matrix A [1], [45]. Let $\mathbf{x}\in\mathbb{R}^N$ be a a graph signal. The output graph signal is the length $N$ vector

$$\mathbf{y}=\mathbf{h}*_v\mathbf{x}=\sum_{\ell=1}^{L}h_\ell\,\mathrm{A}^\ell\,\mathbf{x}. \tag{7}$$

The filter coefficients $\mathbf{h}=\begin{bmatrix}h_1 & h_2 & \cdots & h_L\end{bmatrix}$ are usually fixed and designed based on graph filter banks and graph wavelets [2]. Note that in (7), we remove the 0th order of the graph shift because an identity mapping can trivially produce an output that is the same as the input and does not contribute any denoising effect.

Some variations also consider the vertex-variant graph convolution [49], where each filter coefficient $h_\ell$ is expanded to be a vector, and the edge-variant graph convolution [50], [51], where each filter coefficient $h_\ell$ is expanded to be a matrix. In this paper, we consider (7) as our default graph convolution in any non-neural-network-based model. For example, in the general graph signal denoising problem (2), the graph convolution follows the definition in (7). We will use $\nabla\mathbf{h}*_v\mathbf{x}=\mathbf{h}*_v=\sum_{\ell=1}^{L}h_\ell\,\mathrm{A}^\ell$ to denote the derivative of $\mathbf{h}*_v\mathbf{x}$, which will be used in Section IV.

### B. Graph Convolution in Neural Networks

As one of the most successful neural network models, convolution neural networks (CNNs) use a sequence of trainable convolution operations to extract deep features from input data. The convolution in CNNs follows the same spirit of conventional convolution. At the same time, it allows feature learning in a high-dimensional space, which is one of the most important characteristics of CNNs [26]. A convolution operator usually

contains a large number of trainable parameters and allows for multiple-channel inputs and outputs.

Let $\mathrm{X} = \begin{bmatrix}\mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \dots & \mathbf{x}^{(K)}\end{bmatrix} \in \mathbb{R}^{N\times K}$ be a $K$-channel signal; in other words, there are $K$ features at each time stamp. Let a three-mode tensor $\mathbb{H} \in \mathbb{R}^{L\times K\times K'}$ be a collection of trainable filter coefficients that takes a $K$-channel signal as input and outputs a $K'$-channel signal. Each convolution layer operates as $\mathrm{Y} = \mathbb{H} * \mathrm{X} \in \mathbb{R}^{N\times K'}$, with $k'$th output channel

$$\mathbf{y}^{(k')} = \sum_{\ell=1}^{L}\sum_{k=1}^{K} \mathbb{H}_{\ell,k,k'}\, \mathrm{C}^{\ell}\, \mathbf{x}^{(k)}, \tag{8}$$

where $\ell$ is the index of the filter length, $k$ is the index of the input channel, and $k'$ is the index of the output channel. Each element in $\mathbb{H}$ is trainable and is updated in an end-to-end learning process. This is the standard convolution operation that is widely used in many applications, such as speech recognition and computer vision [26]. Comparing to conventional convolution in signal processing, (8) introduces trainable parameters and the choices of $K, K'$ allow feature learning in high-dimensional spaces.

We can extend trainable convolution to the graph domain by replacing the cyclic-permutation matrix with the graph adjacency matrix. Similar generalizations have been explored in [52]. Define $\mathrm{X} = \begin{bmatrix}\mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \dots & \mathbf{x}^{(K)}\end{bmatrix} \in \mathbb{R}^{N\times K}$ as a $K$-channel graph signal; in other words, there are $K$ features at each vertex. Let a three-mode tensor $\mathbb{H} \in \mathbb{R}^{L\times K\times K'}$ be a collection of trainable graph filter coefficients that takes a $K$-channel graph signal as input and outputs a $K'$-channel graph signal. A trainable graph convolution is $\mathrm{Y} = \mathbb{H} *_w \mathrm{X}$, with $k'$th output channel

$$\mathbf{y}^{(k')} = \sum_{\ell=1}^{L}\sum_{k=1}^{K} \mathbb{H}_{\ell,k,k'}\, \mathrm{A}^{\ell}\, \mathbf{x}^{(k)}, \tag{9}$$

where the response Y is a $N \times K'$ matrix, $\mathbf{y}^{(k')}$ is the $k'$th column of Y and $\mathbb{H}_{\ell,k,k'}$ is trained during learning. We use the symbol $*_w$, instead of $*_v$, to emphasize that the filter coefficients in the proposed graph convolution (9) are trained in end-to-end learning while the filter coefficients in (7) are manually designed.

Graph convolution (9) is analogous to conventional convolution (8); that is, the output signal coefficient at each vertex is a weighted average of the signal coefficients at neighboring vertices. At the same time, (9) is a multi-channel extension of (7): when $K = K' = 1$, (9) degenerates to (7). To make the notation consistent, when $K = K' = 1$, we still use $\mathbf{y} = \mathbb{H} *_w \mathbf{x} = \sum_{\ell=1}^{L} \mathbb{H}_{\ell,1,1}\, \mathrm{A}^{\ell}\, \mathbf{x}$, where $\mathbb{H} \in \mathbb{R}^{L\times 1\times 1}$. An equivalent representation of (9) is,

$$\mathrm{Y} = \sum_{\ell=1}^{L} \mathrm{A}^{\ell}\, \mathrm{X}\, \mathrm{H}^{(\ell)},$$

where $\mathrm{H}^{(\ell)}$ is a $K \times K'$ trainable matrix with $\mathrm{H}^{(\ell)}_{k,k'} = \mathbb{H}_{\ell,k,k'}$. Through multiplying with $\mathrm{H}^{(\ell)}$, we transform X to a high-dimensional feature space. After that, we diffuse the new features over the vertex domain according to the graph adjacency matrix A. A special case is when $L = 1$, in which case (9) degenerates to

$$\mathrm{Y} = \mathrm{A}\,\mathrm{X}\,\mathrm{H}\,. \tag{10}$$

This graph convolution is actively used in semi-supervised vertex classification [8].

### C. Weight-Sharing Mechanism

Previously, we obtained graph convolution through a mathematical analogy with standard convolution. However, the definitions of the neighborhoods are clearly different in the conventional convolution (8) and graph convolution (9). For a time-series, each shift order $\ell$ introduces one neighbor for each time stamp. Given a cyclic convolution of length $L$, each time stamp has $L$ distinct neighbors and is associated with $L$ corresponding filter coefficients. On the other hand, for a graph signal, each graph shift order $\ell$ might introduce zero, one or multiple neighbors for each vertex. The number of neighbors depends on the local graph structure. Given a graph convolution of length $L$, those $L$ filter coefficients are insufficient to reflect distinct weights for all the neighbors. This difference in the neighborhood definition distinguishes (8) and (9) since a vertex cannot adjust the contribution from each of its neighbors individually. Here we propose a new graph convolution to fill this gap.

To make the graph convolution more flexible and powerful, we consider updating the edge weights in the given graph adjacency matrix. In this way, each vertex will have different impact on its neighbors, just like in conventional convolution. A straightforward approach is to introduce a trainable mask matrix $\Psi^{(\ell,k,k')} \in \mathbb{R}^{N\times N}$, expanding a single filter coefficient $\mathbb{H}_{\ell,k,k'}$ in (9) to a matrix of coefficients [51]. We then use $\Psi^{(\ell,k,k')} \odot \mathrm{A}^{\ell}$ to replace $\mathbb{H}_{\ell,k,k'}\, \mathrm{A}^{\ell}$ in (9), so that we make each edge weight trainable. However, when the graph size, $N$, is large, training $O(N^2)$ parameters in $\Psi^{(\ell,k,k')}$ is computationally difficult.

To reduce the number of training parameters, we allow all the edges to share the same set of weights, which is similar to the weight-sharing mechanism in conventional convolution [26]. We aim to design a kernel function to parameterize each edge weight. For example, for 2D convolution in image processing, the key is to use a local kernel function to map the relative difference between two pixel coordinates to a weight. For graphs, we assign a coordinate to each vertex and use a local kernel function to map the relative difference between two vertex coordinates to an edge weight.

The vertex coordinates can be obtained through the graph Fourier transform [45]. Let the eigendecomposition of the graph adjacency matrix be $\mathrm{A} = \mathrm{V}\,\Lambda\,\mathrm{V}^T$, where $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_N)$ is a diagonal matrix of $N$ eigenvalues and V is the matrix of corresponding eigenvectors. The eigenvalues of A represent the *graph frequencies* and the eigenvectors form the *graph Fourier basis*. The coordinate of the $i$th vertex is the row vector of the truncated graph Fourier basis, $\mathbf{p}_i = \begin{bmatrix}\mathrm{V}_{i,1} & \mathrm{V}_{i,2} & \dots & \mathrm{V}_{i,p}\end{bmatrix}^T \in \mathbb{R}^p$, where $p \le N$ is a hyperparameter. Through the graph Fourier transform, we map the information of each vertex from the graph vertex domain to the graph spectral domain.

Next, we assume that the edge weight between the $i$th and the $j$th vertices is parameterized by a kernel function:

$$\Psi_{i,j} = \psi_w\left([\mathbf{p}_j - \mathbf{p}_i]\right) \in \mathbb{R}, \tag{11}$$

where $\psi_w(\cdot)$ is a trainable function, which can be implemented by a multilayer perceptron (MLP) [26]. Given a pair of vertex coordinates, we convert their relative coordinate difference to a scalar that reflects the corresponding edge weight. In 2D convolution for images, the convolution kernel function is independent of the absolute pixel coordinates. Specifically, when we set the origin of the kernel function to a pixel, the weight from each of the pixel's neighbors depends on the relative coordinate difference between a neighboring pixel and the origin. Similarly, here we use the relative coordinate difference as the input because it allows the kernel function $\psi_w(\cdot)$ to be irreverent to the exact coordinate and to be applied to arbitrary edges.

Note that a few previous works also consider learning edge weights. For example, EdgeNet considers each edge weight as an independent trainable parameter [51]. However, the number of trainable parameters depends on the graph size, which is computationally expensive. Graph attention networks learn edge weights through the attention mechanism. Each edge weight is parameterized by a kernel function, whose inputs are graph signals [30]. Here we consider a different parameterization: the input of a kernel function is the difference between a pair of vertex coordinates, which relies on the graph structure and is independent of the graph signals. This approach leverages the graph spectral information, which fuses both global and local information on graphs. The number of trainable parameters depends on the kernel function and is independent of the graph size.

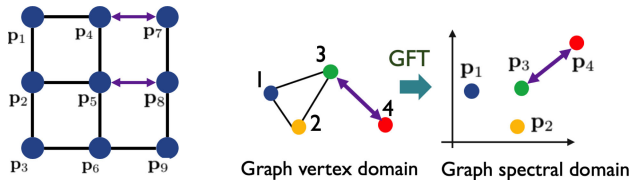
### D. Edge-Weight-Sharing Graph Convolution

We finally apply the convolution kernel function (11) and propose the *edge-weight-sharing graph convolution (EWS-GC)* as $\mathrm{Y} = \mathbb{H} *_a \mathrm{X}$ with $k'$th output channel

$$\mathbf{y}^{(k')} = \sum_{\ell=1}^{L} \sum_{k=1}^{K} \left( \Psi^{(\ell,k,k')} \odot \mathrm{A}^{\ell} \right) \mathbf{x}^{(k)} \in \mathbb{R}^N, \qquad (12)$$

where the response Y is a $N \times K'$ matrix, $\Psi^{(\ell,k,k')} \in \mathbb{R}^{N \times N}$ is an edge-weight matrix whose elements are trainable and follow from (11). We use the symbol $*_a$, instead of $*_w$, to emphasize that (9) works with trainable filter coefficients, yet fixed edge weights; while in (12), edge weights are included in graph filter coefficients, which are all trainable. We stress that the 0th order of the graph shift has been removed in (12) to avoid the resulting network to learn an identity mapping.

Here the graph filter coefficients form a five-mode tensor $\mathbb{H} \in \mathbb{R}^{L \times K \times K' \times N \times N}$ with $\mathbb{H}_{\ell,k,k',i,j} = \Psi_{i,j}^{(\ell,k,k')}$, where $L$ is the filter length, $K$ is the number of input channels, $K'$ is the number of output channels and $N$ is the number of nodes. An intuitive way to consider this filter tensor is that we stack the graph adjacency matrix $\mathrm{A} \in \mathbb{R}^{N \times N}$ for $LKK'$ times; at each time, we are allowed to vary the nonzero edge weights in A to enrich the filtering ability. To implement this tensor of graph filter coefficients, for an edge that connects the $i$th and the $j$th nodes, we input the difference between two graph vertex coordinates $\mathbf{p}_i \in \mathbb{R}^p$ and $\mathbf{p}_j \in \mathbb{R}^p$, the MLP (11) would map $\mathbf{p}_i - \mathbf{p}_j$ to $LKK'$ graph filter coefficients, whose $(\ell, k, k')$th element is then $\mathbb{H}_{\ell,k,k',i,j}$. Since there are maximally $N^2$ edges, the maximum number of filter coefficients are $LKK'N^2$. Through reshaping, we obtain the tensor $\mathbb{H}$. Note that i) a single MLP is shared by all the edges, which is the essence of the edge-weight-sharing mechanism; ii) the graph structure is usually sparse and we do not need to go through all the $N^2$ entries in the graph adjacency matrix. We only need to compute the entries whose original edge weights in the graph adjacency matrix A are nonzero; and iii) when the number of input and output channels are one; that is, $K = K' = 1$, the number of filter coefficients for each edge is simply the filter length $L$, which is the same with the conventional graph filters (7). Since we preserve the connectivity pattern in the graph adjacency matrix, the proposed edge-weight-sharing graph convolution (12) still relies on the given graph structure to propagate information, but has flexibility to adjust the edge weights.




(a) Pixel coordinates. (b) Vertex coordinates.

Fig. 1. Vertex coordinates and edge weights in images and graphs. Plot (a) shows a 2D image, where each pixel naturally is a coordinate on a 2D lattice. A weight in a 2D convolution is determined by the relative difference between a pair of pixel coordinates. Plot (b) shows a irregular graph, where each vertex can be mapped to a vertex coordinate in the graph spectral domain through the graph Fourier transform (GFT). An edge weight in the proposed graph convolution is determined by the relative difference between a pair of vertex coordinates (11).

The convolution (12) promotes weight sharing as all edge weights adopt the same kernel function with shared training parameters; see Fig. 1. Specifically, we can represent a vertex by its coordinate in the graph spectral domain. Since the vertex coordinate is continuous, the proposed graph convolution kernel function can be mathematically rewritten in the form of a continuous convolution. Let us represent $K$ graph signals as a train of delta functions supported on the graph spectral space; that is, $\mathbf{s}(\mathbf{p}) = \sum_{j=1}^{N} \delta(\mathbf{p} - \mathbf{p}_j)\mathbf{x}_j$, where $\mathbf{x}_j \in \mathbb{R}^K$ and $\mathbf{p}_j \in \mathbb{R}^p$ is the $j$th vertex's $K$ signal coefficients and the vertex coordinate in the graph spectral space, respectively. Let a continuous convolution be $\psi_w(\cdot) : \mathbb{R}^p \to \mathbb{R}$. The response is then

$$\begin{aligned} y(\mathbf{p}) &= \int_{\tau \in \mathcal{N}_{\mathbf{p}}} \psi_w(\mathbf{p} - \tau)\mathbf{s}(\tau) d\tau \\ &= \int_{\tau \in \mathcal{N}_{\mathbf{p}}} \psi_w(\mathbf{p} - \tau) \sum_{j=1}^{N} \delta(\tau - \mathbf{p}_j)\mathbf{x}_j d\tau \\ &= \sum_{j \in \mathcal{N}_{\mathbf{p}}} \psi_w(\mathbf{p} - \mathbf{p}_j)\mathbf{x}_j, \end{aligned}$$

where $\mathcal{N}_{\mathbf{p}}$ is the neighborhood of a vertex coordinate $\mathbf{p}$. For instance, the signal output at the $i$th vertex is $y(\mathbf{p}_i) = \sum_{j=1}^{N} \psi_w(\mathbf{p} - \mathbf{p}_j)\, \mathrm{A}_{i,j}\, \mathbf{x}_j$, where the neighborhood of the $i$th vertex is specified by the graph adjacency matrix. We thus can see that the proposed graph convolution can be reformulated as a continuous convolution in the graph spectral domain. Since a continuous convolution has the shift-invariant property, the

proposed edge-weight-sharing graph convolution naturally carries the weight-sharing mechanism. For example, in Fig. 1, the weight between the 4th and the 7th pixels is equal to the weight between the 5th and the 8th pixels because the relative pixel coordinates are the same; that is, $\psi_w(\mathbf{p}_4 - \mathbf{p}_7) = \psi_w(\mathbf{p}_5 - \mathbf{p}_8)$.

We can also present (12) from another perspective. Let $\mathcal{E}^{(\ell)}$ be the edge set associated with the polynomial of the graph adjacency matrix $\mathrm{A}^\ell$. For an arbitrary edge $e = (v_i, v_j) \in \mathcal{E}^{(\ell)}$, $\delta_e$ is defined as a $N \times N$ indicating matrix whose elements are $(\delta_e)_{i,i'j'} = \mathrm{A}^\ell_{i,i'j'}$ when $e = (v_{i'}, v_{j'})$ and 0, otherwise. The subscript $e$ indicates an edge and $\delta_e$ is a one-hot matrix, only activating the element specified by the edge $e$. This edge is associated with a $K \times K'$ trainable matrix $\mathrm{H}^{(e)}$ with elements $\mathrm{H}^{(e)}_{k,k'} = \mathbb{H}_{\ell,k,k,'i,j}$. The equivalent representation of (12) is[1]

$$\mathrm{Y} = \sum_{\ell=1}^{L} \sum_{e \in \mathcal{E}^{(\ell)}} \delta_e \, \mathrm{X} \, \mathrm{H}^{(e)}, \tag{13}$$

where $\mathrm{A}^\ell = \sum_{e \in \mathcal{E}^{(\ell)}} \delta_e$. The first summation considers all edge sets and the second summation considers all edges. We parse a graph to a collection of edges and the effect of each edge is reflected through the corresponding trainable matrix $\mathrm{H}^{(e)}$.

The edge-weight-sharing graph convolution (12) is a specific type of edge-variant graph convolution, which leverages the weight-sharing mechanism to reduce the number of trainable parameters. Indeed, (9) is a special case of (12) when all elements in each $\Psi^{(\ell,k,k')}$ have the same value. Similarly, (10) is a special case of (9) when the filter length $L = 1$.

We can apply the proposed edge-weight-sharing graph convolution (12) as a substitute to conventional graph filtering in the denoising problem (2). In the next section, we use (12) as a building block for graph neural networks. Its associated parameters will be trained in an end-to-end learning process.

Finally, we consider the permutation-equivariant property of the edge-weight-sharing graph convolution (13). Let $\mathrm{J} \in \mathbb{R}^{N \times N}$ be a permutation matrix. After permutation, a graph adjacency matrix $\mathrm{A} \in \mathbb{R}^{N \times N}$ and a $K$-channel graph signal $\mathrm{X} \in \mathbb{R}^{N \times K}$ become $\mathrm{J}\,\mathrm{A}\,\mathrm{J}^T$ and $\mathrm{J}\,\mathrm{X}$, respectively.

*Theorem 1:* The edge-weight-sharing graph convolution (13) is permutation equivariant. Suppose that a kernel function $\psi_w(\cdot)$ is fixed and given. Then,

$$\mathrm{J}\left(\mathbb{H} *_a \mathrm{X}\right) = \mathbb{H} *_a \left(\mathrm{J}\,\mathrm{X}\right).$$

*Proof:* We first show the effect of permutation on the trainable edge-weight matrix (11). When we permute the graph structure, the vertex coordinates permute accordingly. Therefore, after permutation, a edge-weight matrix $\Psi^{(\ell,k,k')}$ becomes $\mathrm{J}\Psi^{(\ell,k,k')}\mathrm{J}^T$. The $k$th channel of $\mathbb{H} *_a (\mathrm{J}\,\mathrm{X})$ becomes

$$\begin{aligned}
&\left(\mathbb{H} *_a \left(\mathrm{J}\,\mathrm{X}\right)\right)^{(k)} \\
&= \sum_{\ell=1}^{L} \sum_{k=1}^{K} \left( \left( \mathrm{J}\Psi^{(\ell,k,k')}\mathrm{J}^T \right) \odot \left( \mathrm{J}\,\mathrm{A}\,\mathrm{J}^T \right)^\ell \right) \mathrm{J}\mathbf{x}^{(k)} \\
&= \mathrm{J} \sum_{\ell=1}^{L} \sum_{k=1}^{K} \left( \Psi^{(\ell,k,k')} \odot \mathrm{A}^\ell \right) \mathbf{x}^{(k)} \\
&= \left(\mathrm{J}\left(\mathbb{H} *_a \mathrm{X}\right)\right)^{(k)},
\end{aligned}$$

which is the the $k$th channel of $\mathrm{J}(\mathbb{H} *_a \mathrm{X})$. ■

The permutation-equivariant property is important because it ensures that reordering of the vertices will not effect the training of the edge-weight-sharing graph convolution. With the new graph convolution operation, we now propose an interpretable framework for designing graph neural networks.

## IV. Graph Unrolling Networks

In Section II, we mathematically formulated the task of graph signal denoising (2). In this section, we aim to solve (2) through algorithm unrolling and propose a general framework for developing graph unrolling networks. The core strategy is to follow iterative algorithms and then use the trainable edge-weight-sharing graph convolution proposed in Section III to substitute fixed graph filtering. We further consider two specific architectures of graph unrolling networks by unrolling graph sparse coding and graph trend filtering.

### A. General Framework

Consider a general iterative algorithm to solve the graph signal denoising problem (2) based on the half-quadratic splitting algorithm. The basic idea is to perform variable-splitting and then alternating minimization on the penalty function [41], [54].

Introduce two auxiliary variables $\mathbf{y} = \mathrm{P}\,\mathbf{x}$ and $\mathbf{z} = \mathrm{Q}\,\mathbf{s}$ and then reformulate (2) as

$$\begin{aligned}
&\min_{\mathbf{s} \in \mathbb{R}^N} \frac{1}{2} \left\| \mathbf{t} - \mathbf{x} \right\|_2^2 + u(\mathbf{y}) + r(\mathbf{z}), \\
&\text{subject to } \mathbf{x} = \mathbf{h} *_v \mathbf{s}, \;\; \mathbf{y} = \mathrm{P}\,\mathbf{x}, \;\; \mathbf{z} = \mathrm{Q}\,\mathbf{s}.
\end{aligned}$$

The penalty function is

$$\begin{aligned}
J(\mathbf{x},\mathbf{s},\mathbf{y},\mathbf{z}) = \frac{1}{2} \left\| \mathbf{t} - \mathbf{x} \right\|_2^2 + u(\mathbf{y}) + r(\mathbf{z}) + \frac{\mu_1}{2} \left\| \mathbf{x} - \mathbf{h} *_v \mathbf{s} \right\|_2^2 \\
+ \frac{\mu_2}{2} \left\| \mathbf{y} - \mathrm{P}\,\mathbf{x} \right\|_2^2 + \frac{\mu_3}{2} \left\| \mathbf{z} - \mathrm{Q}\,\mathbf{s} \right\|_2^2,
\end{aligned}$$

where $\mu_1, \mu_2, \mu_3$ are appropriate step sizes. We alternately minimize $J$ over $\mathbf{x}, \mathbf{s}, \mathbf{y}, \mathbf{z}$, leading to the following updates:

$$\mathbf{x} \leftarrow \widetilde{\mathrm{P}} \Big( \mu_1 \mathbf{h} *_v \mathbf{s} + \mathbf{t} + \mu_2 \, \mathrm{P}^T \, \mathbf{y} \Big), \tag{14a}$$

$$\mathbf{s} \leftarrow \widetilde{\mathrm{Q}} \Big( \mu_1 \mathbf{h} *_v^T \mathbf{x} + \mu_3 \, \mathrm{Q}^T \, \mathbf{z} \Big), \tag{14b}$$

$$\mathbf{y} \leftarrow \arg\min_{\mathbf{y}} \frac{\mu_2}{2} \left\| \mathbf{y} - \mathrm{P}\,\mathbf{x} \right\|_2^2 + u(\mathbf{y}), \tag{14c}$$

$$\mathbf{z} \leftarrow \arg\min_{\mathbf{z}} \frac{\mu_3}{2} \left\| \mathbf{z} - \mathrm{Q}\,\mathbf{s} \right\|_2^2 + r(\mathbf{z}), \tag{14d}$$

where

$$\widetilde{\mathrm{P}} = \left( \mathrm{I} + \mu_1 \, \mathrm{I} + \mu_2 \, \mathrm{P}^T \, \mathrm{P} \right)^{-1}$$

and

$$\widetilde{\mathrm{Q}} = \left( \mu_1 \sum_{\ell'=1}^{L} h_{\ell'} \, \mathrm{A}^{\ell'} \sum_{\ell=1}^{L} h_\ell \, \mathrm{A}^\ell + \mu_3 \, \mathrm{Q}^T \, \mathrm{Q} \right)^{-1}.$$

[1] Equation (13) suggests a randomized implementation of the edge-weight-sharing graph convolution. Instead of using the entire edge sets, we randomly sample a subset of edges and approximate the exact value of (13) as $\mathrm{Y} \approx \sum_{e \in \mathcal{M}} \delta_e \, \mathrm{X} \, \mathrm{H}^{(e)}$. The edge set $\mathcal{M} \subset \mathcal{E}^{(1)} \cup \mathcal{E}^{(2)} \cup \cdots \cup \mathcal{E}^{(L)}$ is obtained through edge sampling, which can be implemented via random walks [53].

Intuitively, (14a) denoises by merging information from the original measurements $\mathbf{t}$, filtered signals $\mathbf{h} *_v \mathbf{s}$ and the auxiliary variable $\mathbf{y}$; (14b) generates a base graph signal through graph deconvolution; and (14c) and (14d) solve two proximal functions with regularization $u(\cdot)$ and $r(\cdot)$, respectively.

In practice, the filter generating graph signals in (14) might be unknown. Instead of solving the original optimization problem exactly, our unrolling network leverages the learning ability to capture complicated filters. To unroll the iteration steps (14), we consider two major substitutions. First, we replace the fixed graph convolution by the trainable edge-weight-sharing graph convolution (12). Second, we replace the sub-optimization problems in (14c) and (14d) by a trainable neural network. Note that because of the above substitutions, while (14) inspires (15), (15) might not guarantee the optimal solution to (2). A unrolling layer in our graph unrolling network is then,

$$\mathbf{x} \leftarrow \mathbb{A} *_a \mathbf{s} + \mathbb{B} *_a \mathbf{t} + \mathbb{C} *_a \left(\mathrm{P}^T \mathbf{y}\right), \tag{15a}$$

$$\mathbf{s} \leftarrow \mathbb{D} *_a \mathbf{x} + \mathbb{E} *_a \left(\mathrm{Q}^T \mathbf{z}\right), \tag{15b}$$

$$\mathbf{y} \leftarrow \mathrm{NN}_u \left(\mathrm{P}\,\mathbf{x}\right), \tag{15c}$$

$$\mathbf{z} \leftarrow \mathrm{NN}_r \left(\mathrm{Q}\,\mathbf{s}\right), \tag{15d}$$

where $\mathbb{A}*_a$, $\mathbb{B}*_a$, $\mathbb{C}*_a$, $\mathbb{D}*_a$, and $\mathbb{E}*_a$ are individual edge-weight-sharing graph convolutions with filter coefficients that are trainable parameters, and $\mathrm{NN}_u(\cdot)$ and $\mathrm{NN}_r(\cdot)$ are two neural networks, which involve trainable parameters. Intuitively, (15a) and (15b) are neural-network implementations of (14a) and (14b), respectively, replacing fixed graph convolutions $\mathbf{h}*_v$ by trainable edge-weight-sharing graph convolutions (12); and (15c) and (15d) are neural-network implementations of the proximal functions (14c) and (14d), respectively, using neural networks to solve sub-optimization problems; see similar substitutions in [35], [39], [41], [42]. The implementations of (15c) and (15d) depend on specific regularization terms, $u(\cdot)$ and $r(\cdot)$. For some $u(\cdot), r(\cdot)$, we might end up with an analytical form for (15c) and (15d). We will show two examples in Sections IV-B and IV-C.

One characteristic of neural networks is to allow feature learning in a high-dimensional space. Instead of sticking to a single channel, we can easily extend (15) to handle multiple input noisy graph signals and enable multiple-channel feature learning. The corresponding $b$th unrolling layer of multi-channel graph signals is

$$\begin{aligned}
\mathrm{X}^{(b)} &\leftarrow \mathbb{A} *_a \mathrm{S}^{(b-1)} + \mathbb{B} *_a \mathrm{T} + \mathbb{C} *_a \left(\mathrm{P}^T \mathrm{Y}^{(b-1)}\right), \\
\mathrm{S}^{(b)} &\leftarrow \mathbb{D} *_a \mathrm{X}^{(b)} + \mathbb{E} *_a \left(\mathrm{Q}^T \mathrm{Z}^{(b-1)}\right), \\
\mathrm{Y}^{(b)} &\leftarrow \mathrm{NN}_u \left(\mathrm{P}\,\mathrm{X}^{(b)}\right), \\
\mathrm{Z}^{(b)} &\leftarrow \mathrm{NN}_r \left(\mathrm{Q}\,\mathrm{S}^{(b)}\right),
\end{aligned} \tag{16}$$

where $\mathrm{T} \in \mathbb{R}^{N \times K}$ is a matrix of $K$ noisy graph signals, $\mathrm{Z}^{(b)} \in \mathbb{R}^{N \times d^{(b)}}$ is the intermediate feature matrix, and $\mathrm{S}^{(b)} \in \mathbb{R}^{N \times D^{(b)}}$ is the output matrix of the $b$th computational block. The feature dimensions $d^{(b)}, D^{(b)}$ are hyperparameters of the network; see a graph unrolling layer in Fig. 2.

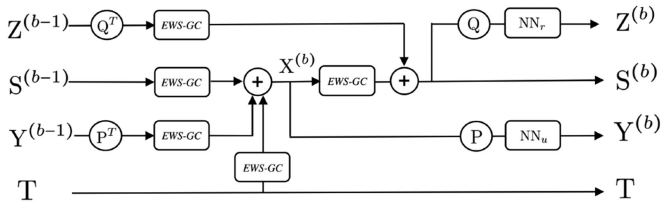



Fig. 2. A generic graph unrolling layer for graph signal denoising (16), which is one computational block in a graph unrolling network. Given the raw measurement T, the proposed unrolling layer updates Z, S, and Y.

To build a complete network architecture, we initialize $\mathrm{Z}^{(0)}, \mathrm{S}^{(0)}, \mathrm{Y}^{(0)}$ to be all-zero matrices and sequentially stack $B$ unrolling layers (16). This is hypothetically equivalent to running the iteration steps (14) for $B$ times. Through optimizing trainable parameters in edge-weight-sharing graph convolutions and two sub-neural-networks, we obtain the denoised output $\widehat{\mathrm{X}} = \mathrm{X}^{(B)}$.

Here the trainable parameters come from two parts, including filter coefficients in each edge-weight-sharing graph convolution and the parameters in the neural networks (15c) and (15d). Through optimizing those parameters, we can capture complicated priors in the original graph signals in a data-driven manner. To train those parameters, we consider the loss function

$$\mathrm{loss} = \left\| f\left(\mathrm{T}\right) - \mathrm{T} \right\|_F^2 = \left\| \widehat{\mathrm{X}} - \mathrm{T} \right\|_F^2, \tag{17}$$

where $\|\cdot\|_F$ is the Frobenius norm, $\widehat{\mathrm{X}}$ is the output of the proposed network $f(\cdot)$, and T are the original measurements. We then use the stochastic gradient descent to minimize the loss and optimize this network [26]. The noisy measurement T is used as both input and supervision of the network.

Hypothetically, the loss could be zero when a neural network is trained to be an identity mapping. In other words, the denoised output is the same as the input noisy measurements. However, this does not happen since the building block of a graph unrolling network is an edge-weight-sharing graph convolution, where the 0th order of the graph shift has been removed. Furthermore, this convolution leverages irregular graph structures and injects implicit graph regularization to the network architecture and the overall optimization problem.

Our algorithm unrolling here is rooted in the half-quadratic splitting algorithm. In practice, our optimization problem can be solved using various alternative iterative algorithms, which may lead to distinct network architectures. No matter what iterative algorithm is used, the core strategy is to follow the iterative steps and use a trainable edge-weight-sharing graph convolution to substitute fixed graph filtering. We call a network architecture that follows this strategy a *graph unrolling network (GUN)*.

Compared to conventional graph signal denoising algorithms [11], [12], the proposed GUN is able to learn a variety of complicated signal priors from given graph signals by leveraging the learning ability of deep neural networks. We emphasize that our signal model assumes that the graph signal is generated from graph filtering, but empirically this graph filter is trained through the end-to-end learning. Compared to many generic graph neural

networks [8], the proposed GUN is interpretable. Here interpretability means that a method is developed by modeling the physical processes underlying the problem or capturing prior domain knowledge. The proposed general iterative algorithm of solving the graph signal denoising problem (14) is interpretable because i) the iteration procedure minimizes an explicit objective function; and ii) the objective function models the potential signal priors through graph filtering and graph regularization terms. Similarly to many other unrolling methods, our network layers naturally inherit interpretability from the iteration procedure by following analytical iterative steps. We unroll an iterative algorithm for solving (2) into a graph neural network by mapping each iteration into a single network layer and stacking multiple layers together. In this manner, the proposed GUN can be naturally interpreted as a parameter optimized algorithm; see other examples of interpretable unrolling networks in the review paper [38].

In the following, we present two special cases of GUN, which are obtained through unrolling graph sparse coding and graph trend filtering, respectively. Graph sparse coding is a typical graph-dictionary-based denoising algorithm, where we first design a graph dictionary based on a series of graph filters and then select a few elementary graph signals from a graph dictionary to approximate a noisy graph signal [2], [20], [21]. The unrolling version of graph sparse coding essentially combines these two steps in an end-to-end learning process and uses an edge-weight-sharing graph convolution to substitute the predesigned graph dictionary; we call this architecture a *graph unrolling sparse coding (GUSC)*. On the other hand, graph trend filtering is a typical graph-regularization-based denoising algorithm, where we first formulate an optimization problem with graph total variation and then solve this optimization problem to denoise graph signals [14], [15]. The unrolling version of graph trend filtering uses a trainable edge-weight-sharing graph convolution to provide an implicit graph regularization and uses end-to-end learning to optimize the trainable parameters; this resulting architecture is called a *graph unrolling trend filtering (GUTF)*.

Comparing these two methods, graph trend filtering is designed for piecewise-constant and piecewise-smooth graph signals, so that GUTF is more regularized. On the other hand, graph sparse coding works for a broader class of graph signals, resulting in GUSC being more general, but typically requires more training data. In the experiments, we will see that GUTF achieves better denoising performance than GUSC for simulated data, including smooth graph signals, piecewise-constant graph signals, and piecewise-smooth graph signals. When the number of graph signals increases, the gap between GUTF and GUSC decreases. On the other hand, GUSC achieves better denoising performance than GUTF for real-world data, which has more complicated structure than simulated data.

### B. Graph Sparse Coding

As discussed in Section II-A1, graph sparse coding (3) considers reconstructing noiseless graph signals through graph filtering with sparse base graph signals. In this setting, $\mathbf{x} = \mathbf{h} *_v \mathbf{s} = \sum_{\ell=1}^{L} h_\ell \, \mathrm{A}^\ell \, \mathbf{s}$, $u(\cdot) = 0, r(\cdot) = \alpha \| \cdot \|_1$ and $\mathrm{P} = \mathrm{Q} = \mathrm{I}$.

**Algorithm 1:** Graph Unrolling Sparse Coding (GUSC).

**Input** T matrix of measurements
A graph adjacency matrix
$B$ number of network layers
$\alpha$ hyperparameter
**Output** $\widehat{\mathrm{X}}$ matrix of denoised graph signals
**Function GSC**$(\mathrm{T}, \mathrm{A}, B, \alpha)$
$\mathrm{S}^{(0)} \leftarrow \mathbf{0}$
$\{\mathbf{p}_i\}_{i \in \mathcal{V}} \leftarrow$ eigendecomposition of A
for $b = 1 : B$
$\mathbb{A}, \mathbb{B}, \mathbb{D}, \mathbb{E}_{\ell,k,k,'i,j} \leftarrow \mathrm{MLP}([\mathbf{p}_j - \mathbf{p}_i])$
$\mathrm{X}^{(b)} \leftarrow \mathbb{A} *_a \mathrm{S}^{(b-1)} + \mathbb{B} *_a \mathrm{T}$,
$\mathrm{S}^{(b)} \leftarrow \mathbb{D} *_a \mathrm{X}^{(b)} + \mathbb{E} *_a \mathrm{Z}^{(b-1)}$,
$\mathrm{Z}^{(b)} \leftarrow S_\alpha(\mathrm{S}^{(b)})$
end
$\mathbb{H}_{\ell,k,k,'i,j} \leftarrow \mathrm{MLP}([\mathbf{p}_j - \mathbf{p}_i])$
$\widehat{\mathrm{X}} \leftarrow \mathbb{H} *_a \mathrm{S}^{(B)}$
minimize $\|\widehat{\mathrm{X}} - \mathrm{T}\|_F^2$ and update all the parameters
**return** $\widehat{\mathrm{X}}$

We can plug in those specifications to (15) and obtain a customized graph unrolling network. We consider two modifications. First, we remove the terms related to $\mathbf{y}$ because $u(\mathbf{y}) = 0$ and $\mathbf{y}$ should not effect optimization anymore. Second, we replace (15d) by a soft-thresholding function because it is the analytical solution of (14d) [55]. We finally obtain the $b$th unrolling layer customized for graph sparse coding as

$$\begin{aligned} \mathrm{X}^{(b)} &\leftarrow \mathbb{A} *_a \mathrm{S}^{(b-1)} + \mathbb{B} *_a \mathrm{T}, \\ \mathrm{S}^{(b)} &\leftarrow \mathbb{D} *_a \mathrm{X}^{(b)} + \mathbb{E} *_a \mathrm{Z}^{(b-1)}, \\ \mathrm{Z}^{(b)} &\leftarrow S_\alpha \left( \mathrm{S}^{(b)} \right), \end{aligned}$$

where $\alpha$ is a hyperparameter and $S_\cdot(\cdot)$ is a soft-thresholding function,

$$[S_\alpha(\mathbf{x})]_i = \begin{cases} x_i - \alpha, & \text{if } x_i > -\alpha, \\ 0, & \text{if } -\alpha \le x_i \le \alpha, \\ x_i + \alpha, & \text{if } x_i < -\alpha. \end{cases}$$

All the training parameters are involved in the edge-weight-sharing graph convolutions, $\mathbb{A}*_a, \mathbb{B}*_a, \mathbb{C}*_a, \mathbb{D}*_a$ and $\mathbb{E}*_a$. The training paradigm follows the general graph unrolling network; see its overall implemetation in Algorithm 1. The hyperparameter $\alpha$ in the soft-thresholding function could be trainable. In the experiments, we find that the performance of a fixed $\alpha$ is slightly better than a trainable $\alpha$; see Section V-B.

### C. Graph Trend Filtering

As discussed in Section II-A3, graph trend filtering (5) introduces a graph total variation term to regularize the sparsity of the first-order difference of a graph signal. In this case, $\mathbf{x} = \mathbf{h} *_v \mathbf{s} = \mathbf{s}$, $u(\cdot) = \alpha \| \cdot \|_1$, $r(\cdot) = 0$, $\mathrm{P} = \Delta$ and $\mathrm{Q} = \mathrm{I}$.

Plugging these specifications into (15) leads to a customized graph unrolling network. We consider three modifications. First, we remove the terms related to $\mathbf{z}$ because $r(\mathbf{z}) = 0$. Second, we

**Algorithm 2:** Graph Unrolling Trend Filtering (GUTF).

**Input** $\mathrm{T}$ matrix of measurements
$\mathrm{A}$ graph adjacency matrix
$B$ number of network layers
$\alpha$ hyperparameter
**Output** $\widehat{\mathrm{X}}$ matrix of denoised graph signals
**Function GTF**$(\mathrm{T}, \mathrm{A}, B, \alpha)$
$\mathrm{X}^{(0)} \leftarrow \mathbf{0}$
Obtain $\Delta$ from A via (6)
$\{\mathbf{p}_i\}_{i\in\mathcal{V}} \leftarrow$ eigendecomposition of A
for $b = 1 : B$
$\mathbb{B}_{\ell,k,k,'i,j}, \mathbb{C}_{\ell,k,k,'i,j} \leftarrow \mathrm{MLP}([\mathbf{p}_j - \mathbf{p}_i])$
$\mathrm{Y}^{(b)} \leftarrow S_\alpha(\Delta\, \mathrm{X}^{(b-1)})$
$\mathrm{X}^{(b)} \leftarrow \mathbb{B} *_a \mathrm{T} + \mathbb{C} *_a (\Delta^T\, \mathrm{Y}^{(b)})$,
end
$\widehat{\mathrm{X}} \leftarrow \mathrm{X}^{(B)}$
minimize $\|\widehat{\mathrm{X}} - \mathrm{T}\|_F^2$ and update all the weights
**return** $\widehat{\mathrm{X}}$

remove the terms related to $\mathbf{s}$ because $\mathbf{x} = \mathbf{s}$ and there is no need to update both. Third, we replace (15c) by a soft-thresholding function which is the analytical solution of (14c) [55]. We finally obtain the $b$th unrolling layer customized for graph trend filtering to be

$$\mathrm{X}^{(b)} \leftarrow \mathbb{B} *_a \mathrm{T} + \mathbb{C} *_a \left(\Delta^T\, \mathrm{Y}^{(b-1)}\right),$$

$$\mathrm{Y}^{(b)} \leftarrow S_\alpha\left(\Delta\, \mathrm{X}^{(b)}\right),$$

where $S_\cdot(\cdot)$ is a soft-thresholding function. All the training parameters are involved in the edge-weight-sharing graph convolutions, $\mathbb{B}*_a$ and $\mathbb{C}*_a$. The training paradigm follows the general graph unrolling network; see its overall implemetation in Algorithm 2.

Comparing GUTF and GUSC, both follow from the general graph unrolling framework and are based on the proposed edge-weight-sharing graph convolution. The main difference is that GUTF involves a vertex-edge dual representation, where the vertex-based features and edge-based features are converted through the graph incident matrix $\Delta$. This design is potentially better in capturing fast transitions over edges, leading to improved denoising performance on piecewise-constant graph signals. GUSC heavily relies on the learning ability of the edge-weight-sharing graph convolution, which is more general, but needs more training data than GUTF.

## V. Experimental Results

In this section, we evaluate the proposed graph unrolling networks on denoising both simulated and real-world graph signals. Our experiments show that the proposed graph unrolling networks consistently achieve better denoising performance than conventional graph signal denoising algorithms and state-of-the-art graph neural networks on both simulated and real-world graph signals under Gaussian noise, mixture noise and Bernoulli noise.

### A. Experimental Setup

**Configurations.** For GUSC and GUTF, we set the number of epochs for stochastic gradient descent to be 5000, the number of network layers $B = 1$, feature dimension $d^{(b)} = D^{(b)} = 64$, the threshold in the soft-thresholding function $\alpha = 0.05$ in all the cases. To make a fair comparison, we use the same network setting and training paradigm for graph unrolling networks to train other networks.

**Baselines.** We consider three classes of competitive denoising algorithms: graph-regularized optimizations, graph filter banks and neural networks. For graph-regularized optimizations, we select graph Laplacian denoising (GLD) [11] and graph trend filtering (GTF) [14]. Both algorithms introduce graph-regularization terms to the optimization problem. For graph filter banks, we consider graph Fourier transform (GFT) [11], spectral graph wavelet transform (SGWT) [2], graph quadrature-mirror-filters (QMF) [24] and critically sampled filter banks (CSFB) [56]. In each case, we obtain the corresponding graph dictionary and use basis pursuit denoising [55] to reconstruct graph signals from noisy inputs. As competitive neural networks, we consider a multilayer perceptron with three fully-connected layers [26], graph convolution networks (GCN) with three graph convolution layers [8], graph attention networks (GAT) with one graph attention layer [30] and graph autoencoder (GAE) with three graph convolution layers and one kron-reduction pooling layer [57]. In many supervised-learning tasks, previous works realized that deep graph neural networks with too many layers can suffer from oversmoothing and hurt the overall performance [32], [58]. We also find that more layers do not lead to better denoising performance even with residual connections. We tune hyperparameters for each denoising algorithm and report the best performances.

**Graph signals.** We consider three types of simulated graph signals as well as three types of real-world graph signals. For simulations, we consider smooth, piecewise-constant and piecewise-smooth graph signals; for real-world scenarios, we consider temperature data supported on the U.S weather stations, traffic data based on the NYC street networks and community memberships based on citation networks. The details will be elaborated in each case.

**Noise models.** We consider three types of noise to validate the denoising algorithms: Gaussian noise, the mixture noise and Bernoulli noise. In the measurement model (1), we use a length-$N$ vector $\mathbf{e}$ to denote noise. For Gaussian noise, each element of $\mathbf{e}$ follows a Gaussian distribution with zero mean; that is, $\mathbf{e}_i \sim \mathcal{N}(0, \sigma^2)$. The default standard deviation is $\sigma = 0.5$. For the mixure noise, each element of $\mathbf{e}$ follows a mixture of Gaussian distribution and Laplace distribution; that is, $\mathbf{e}_i \sim \mathcal{N}(0, \sigma^2) + Laplace(0, b)$. By default, we set $\sigma = 0.2, b = 0.2$. For binary graph signals, we consider adding Bernoulli noise [59]; that is, we randomly select a subset of vertices and flip the associated binary values. Note that (i) the proposed graph unrolling network is not designed for this noise model, but surprisingly, it still performs well; (ii) we change the loss function (17) to the cross-entropy loss during training; and (iii) this denoising task is essentially a classification task, identifying whether the binary value at each vertex is flipped.

TABLE I
DENOISING ERROR OF THREE TYPES OF SIMULATED GRAPH SIGNALS WITH GAUSSIAN NOISE

| METHOD | SMOOTH | | | | PIECEWISE-CONSTANT | | | | PIECEWISE-SMOOTH | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 10 | 100 | 1000 | 1 | 10 | 100 | 1000 | 1 | 10 | 100 | 1000 |
| BASELINE | 0.563 | 0.517 | 0.498 | 0.5 | 0.563 | 0.516 | 0.498 | 0.5 | 0.562 | 0.517 | 0.498 | 0.5 |
| GLD | 0.078 | 0.076 | 0.071 | 0.07 | 0.045 | 0.045 | 0.047 | 0.042 | 0.114 | 0.111 | 0.143 | 0.13 |
| GTF | 0.098 | 0.1 | 0.093 | 0.094 | 0.039 | 0.043 | 0.045 | 0.039 | 0.135 | 0.115 | 0.104 | 0.111 |
| GFT | 0.159 | 0.125 | 0.111 | 0.112 | 0.078 | 0.077 | 0.082 | 0.077 | 0.162 | 0.157 | 0.185 | 0.171 |
| SGWT | 0.117 | 0.128 | 0.107 | 0.110 | 0.087 | 0.086 | 0.079 | 0.072 | 0.172 | 0.146 | 0.153 | 0.165 |
| QMF | 0.319 | 0.322 | 0.327 | 0.334 | 0.373 | 0.369 | 0.349 | 0.346 | 0.361 | 0.332 | 0.326 | 0.343 |
| CSFB | 0.106 | 0.101 | 0.069 | 0.075 | 0.101 | 0.094 | 0.104 | 0.097 | 0.173 | 0.163 | 0.231 | 0.197 |
| MLP | 0.373 | 0.189 | 0.079 | 0.032 | 0.182 | 0.137 | 0.04 | 0.014 | 0.335 | 0.209 | 0.089 | 0.037 |
| GCN | 0.067 | 0.058 | 0.039 | 0.037 | 0.048 | 0.039 | 0.028 | 0.024 | 0.102 | 0.094 | 0.118 | 0.074 |
| GAT | 0.062 | 0.057 | 0.032 | 0.028 | **0.034** | 0.05 | **0.023** | 0.018 | 0.095 | 0.076 | 0.045 | 0.033 |
| GUSC | 0.049 | 0.053 | 0.029 | **0.023** | 0.036 | 0.04 | 0.024 | 0.014 | 0.074 | 0.069 | **0.031** | **0.022** |
| GUTF | **0.045** | **0.046** | **0.027** | **0.023** | 0.035 | **0.039** | **0.023** | **0.011** | **0.066** | **0.064** | **0.031** | **0.022** |

**Evaluation metrics.** To evaluate the denoising performance, the default metric is the normalized mean square error (NMSE); that is, $\text{NMSE} = \|\widehat{\mathbf{x}} - \mathbf{x}\|_2^2 / \|\mathbf{x}\|_2^2$, where $\mathbf{x} \in \mathbb{R}^N$ is a noiseless graph signal, and $\widehat{\mathbf{x}}$ is a denoised graph signal. A smaller value of NMSE indicates a better denoising performance. We also consider the normalized mean absolute error (NMAE); that is, $\text{NMAE} = \|\widehat{\mathbf{x}} - \mathbf{x}\|_1 / \|\mathbf{x}\|_1$.

For binary graph signals, we evaluate by the error rate (ER), $\text{ER} = \sum_{i=1}^N \mathbf{1}(x_i \neq \widehat{x}_i)/N$, where $x_i$ and $\widehat{x}_i$ are the $i$th element in $\mathbf{x}, \widehat{\mathbf{x}}$, respectively. A smaller value of ER indicates better denoising performance. We also consider the F1 score, which is the harmonic mean of the precision and recall. A higher value of F1 indicates a better denoising performance.

### *B. Simulation Validation*

**Smooth graph signals.** We simulate a random geometric graph, by generating an undirected graph with 500 vertices randomly sampled from the unit square. Two vertices are connected when their Euclidean distance is less than a threshold. To generate a smooth graph signal, we consider bandlimited graph signals [4]. We conduct the eigendecomposition of the graph Laplacian matrix and ascendingly order the eigenvalues. The first few eigenvectors span a subspace of smooth graph signals, called a bandlimited space [4]. We use a linear combination of the first few eigenvectors to obtain a smooth graph signal; see an illustration in Appendix.

We denoise four different numbers of graph signals: $1, 10, 100$ and 1000. We expect that graph unrolling networks will provide better performances with more samples. Columns $2-5$ in Table I compares the denoising performances of smooth graph signals under Gaussian noise. We see that (i) the proposed two graph unrolling networks significantly outperforms all the other competitive methods. For denoising a single graph signal, GUTF is around 40% better than the standard graph Laplacian denoising; for denoising 1000 graph signal, GUTF is around 70% better than the standard graph Laplacian denoising! The intuition is that by following the analytical iterative steps and using the edge-weight-sharing graph convolution, the proposed unrolling network carries an implicit regularization, allowing it to learn graph signal prior with a few examples; see similar phenomenons in image restoration [60], [61]. (ii) among the conventional graph signal denoising algorithms, graph Laplacian denoising achieves the best performances; (iii) neural-network-based methods overall outperform conventional graph signal denoising algorithms. Surprisingly, even training with a single graph signal, most neural networks still provide excellent denoising performance; and (iv) as standard neural networks, MLP performs poorly when training samples are few and gets better when the number of training samples is increased, which makes sense because MLP does not leverage any graph structure. This shows that we cannot expect an arbitrary neural network without dedicated design to work well for graph signal denoising.

**Piecewise-constant graph signals.** We next simulate piecewise-constant graph signals on a random geometric graph. We first randomly partition the graph into a fixed number of connected and mutually exclusive subgraphs with roughly the same size. Within each subgraph, for each graph signal, we randomly generate a constant value over all vertices in the subgraph. The generated graph signal is piecewise-constant and only changing at the boundary between graph partitions; see an example in Appendix.

Again, we denoise four different numbers of graph signals: $1, 10, 100$ and 1000. Columns $6-9$ Table I compare the denoising performances of piecewise-constant graph signals under Gaussian noise. Similar to smooth graph signals, we see that (i) the proposed two graph unrolling networks still significantly outperform all the other competitive methods; ii) among the conventional graph signal denoising algorithms, graph trend filtering achieves the best performances as its graph regularization promotes piecewise-constant graph signals; and (iii) MLP fails with few training samples.

**Piecewise-smooth graph signals.** We simulate piecewise-smooth graph signals on a random geometric graph. Similar to piecewise-constant signals, we first partition the graph into mutually exclusive subgraphs. Within each subgraph we generate smoothing signals based on the first-$K$ eigenvectors of the subgraph's Laplacian matrix, using the same approach as generating smooth graph signals. The combined signal over the whole graph is piecewise-smooth.

TABLE II
DENOISING ERROR OF REAL-WORLD DATA WITH MIXTURE NOISE (GAUSSIAN AND LAPLACE)

| METRIC | TEMPERATURE NMSE | | TRAFFIC NMSE | | CORA ERROR RATE | | CORA F1 SCORE | |
|---|---|---|---|---|---|---|---|---|
| METHOD | 1 | 365 | 1 | 24 | 1 | 7 | 1 | 7 |
| BASELINE | 0.34 | 0.377 | 0.392 | 0.377 | 0.095 | 0.099 | 0.829 | 0.695 |
| GLD | 0.045 | 0.024 | 0.248 | 0.255 | 0.055 | 0.032 | 0.609 | 0.793 |
| GTF | 0.079 | 0.036 | 0.202 | 0.176 | 0.06 | 0.039 | 0.484 | 0.532 |
| GFT | 0.065 | 0.053 | 0.257 | 0.231 | 0.079 | 0.053 | 0.459 | 0.489 |
| SGWT | 0.069 | 0.11 | 0.184 | 0.162 | 0.074 | 0.073 | 0.551 | 0.569 |
| QMF | 0.26 | 0.31 | **0.18** | 0.185 | 0.087 | 0.087 | 0.51 | 0.512 |
| CSFB | 0.07 | 0.061 | 0.344 | 0.36 | 0.129 | 0.143 | 0.43 | 0.437 |
| MLP | 0.142 | 0.027 | 0.31 | 0.169 | 0.095 | 0.072 | 0.829 | 0.695 |
| GCN | 0.041 | 0.033 | 0.293 | 0.279 | 0.042 | 0.025 | 0.903 | 0.901 |
| GAT | 0.044 | 0.031 | 0.267 | 0.264 | 0.041 | 0.032 | 0.909 | 0.873 |
| GUSC | **0.037** | **0.016** | 0.324 | 0.178 | **0.04** | **0.024** | **0.91** | **0.906** |
| GUTF | 0.053 | 0.019 | 0.266 | **0.158** | 0.041 | 0.03 | 0.906 | 0.885 |

Columns $10-12$ in Table I compare the denoising performances of piecewise-smooth graph signals under Gaussian noise. Similar to smooth graph signals, we see that (i) the proposed two graph unrolling networks still significantly outperform all the other competitive methods; ii) among the conventional graph signal denoising algorithms, graph trend filtering achieves the best performances as its graph regularization promotes piecewise-constant graph signals; and (iii) MLP fails with few training samples.

### C. Real-World Examples

**U.S. temperature data.** We consider 150 weather stations in the United States that record their local temperatures [45]. Each weather station has 365 days of recordings (one recording per day), for a total of 54 750 measurements. The graph representing these weather stations is obtained by measuring the geodesic distance between each pair of weather stations. The vertices are represented by an 8-nearest neighbor graph, in which vertices represent weather stations, and each station is connected to the eight closest weather stations. Each graph signal is the daily temperature values recorded in each weather station. We have 365 graph signals in total. Those graph signals are dependent, yet different: they are smooth over the underlying graph as neighboring weather stations record similar temperatures.

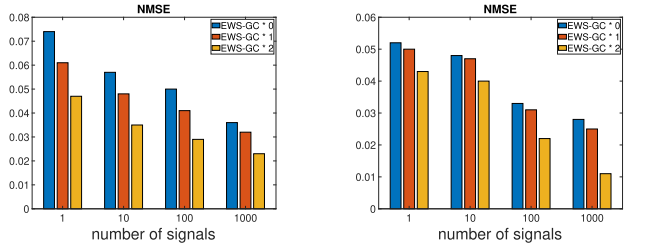
We denoise two different numbers of graph signals: 1 and 365. Columns $2-3$ in Table II compare the denoising performances under the mixture noise. We see that i) the proposed GUSC significantly outperforms all the other competitive methods in terms of NMSE; ii) the proposed GUTF does not work well for a single graph signal, but performs well when more training data is given; and iii) The graph Laplacian denoising achieves the best performance among the conventional graph signal denoising methods. The reason behind that is that the temperature data is smooth over the sensor network and the graph Laplacian-based prior nicely captures the smooth signal prior.

**NYC traffic data.** We consider the taxi-pickup activity in Manhattan on January 1th, 2014. This is the Manhattan street network with 2552 intersections and 3153 road segments. We model each intersection as a vertex and each road segment as an edge. We model the taxi-pickup positions as signals supported on the Manhattan street network. We project each taxi-pickup to its nearest intersection, and then count the number of taxi-pickups at each intersection. Each graph signal is the hourly number of taxi-pickups recorded in each intersection. We consider 24 graph signals for one day in total. Compared to temperature data, the graph signals here are much more complicated. Even two adjacent intersections might not have correlations and exhibit different traffic behaviors.

We denoise two different numbers of graph signals: 1 and 24. Columns $4-5$ in Table II compare the denoising performances under the mixture noise. All the denoising algorithms fail to achieve fine performances. The reason might be that this traffic data is too complicated for a mathematically-designed graph filter to precisely reflect a graph signal prior. In this situation, we see that when more training data is available, the proposed two graph unrolling networks get better performance, reflecting the powerful learning ability to capture complicated priors.

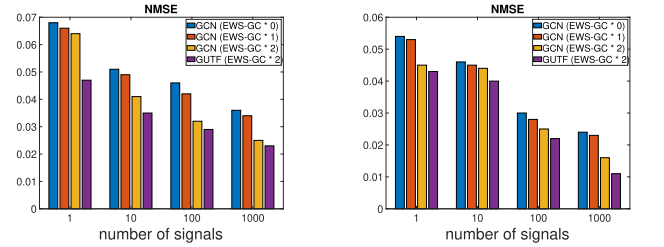
**Cora.** We finally consider a citation network dataset, called Cora [8]. The datasets contain sparse bag-of-words feature vectors for each document and a list of citation links between documents. We treat each citation link as an undirected edge and each document as a class label. The citation network has 2708 nodes and 5429 edges and 7 class labels. We consider 7 class labels as graph signals. We introduce Bernoulli noise and randomly flip 10% of the binary values.

We denoise two different numbers of graph signals: 1 and 7. Columns $6-9$ in Table II compare the denoising performances under Bernoulli noise with two evaluation metrics. For error rates, lower values mean better performances; for F1 scores, higher values mean better performances. We see that i) the proposed GUSC achieves the best performances in terms of both error rates and F1 scores; 2) since zeros appear much more frequently than ones in graph signals, most conventional methods tend to generate zero values everywhere, leading to good error rates, but bad F1 scores.



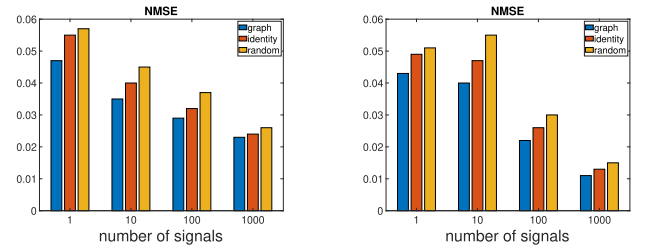

(a) Smooth signals. (b) Piecewise-constant signals.

Fig. 3. Normalized mean square errors of graph unrolling trend filtering with various configurations. The blue bar considers two standard learnable graph convolution; the red bar considers one edge-weight-sharing graph convolutions (EWS-GC) for $\mathbb{B}$ and one standard learnable graph convolution for $\mathbb{C}$; and the yellow bar considers two edge-weight-sharing graph convolutions. We see that the proposed edge-weight-sharing graph convolution works better than the standard learnable graph convolution [8].




(a) Smooth signals. (b) Piecewise-constant signals.

Fig. 4. Normalized mean square errors of graph unrolling trend filtering with various configurations. The blue bar considers the graph convolutional network architecture with two standard learnable graph convolutions; the red bar considers the graph convolutional network architecture with one edge-weight-sharing graph convolutions (EWS-GC) and one standard learnable graph convolutions; the yellow bar considers the graph convolutional network architecture with two edge-weight-sharing graph convolutions (EWS-GC); the purple bar considers the graph unrolling trend filtering architecture with two edge-weight-sharing graph convolutions (EWS-GC). We see that both the proposed unrolling architecture and the proposed edge-weight-sharing graph convolution bring distinct benefits.




(a) Smooth signals. (b) Piecewise-constant signals.

Fig. 5. Normalized mean square errors of the proposed graph unrolling trend filtering with various graph configurations. The blue bar shows the original graph adjacency matrix; the red bar shows the identity matrix and the yellow bar shows the random matrix. The graph does influence the result.

### *D. Ablation Study*

**Effect of edge-weight-sharing graph convolutions.** Here we validate the contribution of each individual edge-weight-sharing graph convolution in graph unrolling trend filtering. Figs. 3 (a) and (b) show the normalized mean square errors of graph unrolling trend filtering with various configurations on smooth and piecewise-constant graph signals, respectively. In the implementation of graph unrolling trend filtering, there are two edge-weight-sharing graph convolutions (EWS-GCs), including $\mathbb{B}$ and $\mathbb{C}$; see Section IV.C. In each plot, we consider three configurations for graph unrolling trend filtering: (i) implementing $\mathbb{B}$ and $\mathbb{C}$ by two standard learnable graph convolution proposed in [8] (blue bar); (ii) implementing $\mathbb{B}$ by the edge-weight-sharing graph convolution and $\mathbb{C}$ by the standard learnable graph convolution (red bar); and (iii) implementing both $\mathbb{B}$ and $\mathbb{C}$ by the edge-weight-sharing graph convolutions (yellow bar). The experimental results show that both edge-weight-sharing graph convolutions bring significant benefits. For the smooth graph signals, $\mathbb{B}$ and $\mathbb{C}$ have similar effects; for the piecewise-smooth graph signals, $\mathbb{C}$ makes more impact than $\mathbb{B}$. This makes sense because the term $\mathbb{C} *_a (\Delta^T \mathrm{Y}^{(b-1)})$ involves the graph incident matrix and implicitly carries the graph regularization term $\|\Delta\|_1$ in the original formulation of graph trend filtering, which is known to favor piecewise-constant graphs signals [14], [15].

To summarize, (i) the proposed edge-weight-sharing graph convolution works better than the standard learnable graph convolution; and (2) the impact of each edge-weight-sharing graph convolution highly depends on the specific properties of graph signals.

**Effect of unrolling architecture.** To understand the gain from the unrolling architecture, we consider the empirical comparison between the standard graph convolutional network (GCN) and the proposed graph unrolling trend filtering (GUTF). In the standard graph convolutional network, we replace the standard learnable graph convolution (GC) by the proposed edge-weight-sharing graph convolution (EWS-GC). In this setting, we could make two comparisons: (i) the GCN architecture with GC vs. the GCN architecture with EWS-GC; and (ii) the GCN architecture with EWS-GC vs. the GUTF architecture with EWS-GC. The first comparison shows the impact of the proposed EWS-GC in the GCN architecture and the second comparison shows the impact of the proposed unrolling architecture.

Figs. 4(a) and (b) show the normalized mean square errors of various graph neural networks on smooth and piecewise-constant graph signals, respectively. We consider four settings: the GCN architecture with two standard GC (blue bar), the GCN architecture with one EWS-GC and one standard GC (red bar), the GCN architecture with two EWS-GC (red bar), the GUTF architecture with two EWS-GC (purple bar). We see that (i) given the same GCN architecture, more EWS-GCs lead to better performances; and (ii) given the same number of EWS-GCs, the GUTF architecture works better than the GCN architecture.

Therefore, we conclude that both the proposed unrolling architecture and the proposed edge-weight-sharing graph convolution bring distinct benefits.

**Effect of graph structure.** Figs. 5(a) and (b) show the normalized mean square errors of the proposed graph unrolling trend filtering with various graph configurations. We consider three graph configurations: original graph adjacency matrix (blue), identity matrix (red) and the random matrix (yellow). We see that the original graph adjacency matrix leads to significantly better performances than the other two configurations, indicating that the graph does influence the performance of the graph unrolling network.

## VI. Conclusion

We propose graph unrolling networks, which is an interpretable neural network framework to denoise single or multiple noisy graph signals. The proposed graph unrolling networks expand algorithm unrolling to the graph domain. As a core component of graph unrolling networks, we propose an edge-weight-sharing graph convolution operation, which parameterizes each edge weight by a trainable kernel function where the trainable parameters are shared by all the edges. This convolution is permutation-equivariant and can flexibly adjust the edge weights to various graph signals. We then consider two specific networks, graph unrolling sparse coding and graph unrolling trend filtering, by unrolling sparse coding and trend filtering, respectively. Through extensive experiments, we show that the suggested methods produce smaller denoising errors than both conventional denoising algorithms and state-of-the-art graph neural networks. Even for denoising a single graph signal, the normalized mean square error of the proposed networks is around 40% and 60% lower than that of graph Laplacian denoising and graph wavelets, respectively, reflecting the advantages of learning from only a few training samples.

## References


[1] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, May 2018.

[2] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmon. Anal.*, vol. 30, pp. 129–150, Mar. 2011.

[3] D. I. Shuman, M. Javad Faraji, and P. Vandergheynst, "A multiscale pyramid transform for graph signals," *IEEE Trans. Signal Process.*, vol. 64, no. 8, pp. 2119–2134, Apr. 2016.

[4] S. Chen, R. Varma, A. Sandryhaila, and J. Kovačević, "Discrete signal processing on graphs: Sampling theory," *IEEE Trans. Signal Process.*, vol. 63, no. 24, pp. 6510–6523, Dec. 2015.

[5] A. Anis, A. Gadde, and A. Ortega, "Efficient sampling set selection for bandlimited graph signals using graph spectral proxies," *IEEE Trans. Signal Process.*, vol. 64, pp. 3775–3789, Jul. 2016.

[6] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 44–63, May 2019.

[7] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017.

[8] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Representations*, Toulon, France, 2017, pp. 24–26

[9] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 146:1–146:12, 2019.

[10] M. Li, S. Chen, Y. Zhao, Y. Zhang, Y. Wang, and Q. Tian, "Dynamic multiscale graph neural networks for 3D skeleton-based human motion prediction," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 211–220.

[11] 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.

[12] S. Chen, A. Sandryhaila, J. M. F. Moura, and J. Kovačević, "Signal denoising on graphs via graph filtering," *Proc. IEEE Global Conf. Signal Inf. Process.*, Dec. 2014, pp. 872–876.

[13] M. Vetterli, J. Kovačević, and V. K. Goyal, *Foundations of Signal Processing*. Cambridge, U.K.:, Cambridge Univ. Press, 2014.

[14] Y-X. Wang, J. Sharpnack, A. J. Smola, and R. J. Tibshirani, "Trend filtering on graphs," *J. Mach. Learn. Res.*, vol. 17, pp. 105:1–105:41, 2016.

[15] R. Varma, H. Lee, J. Kovačević, and Y. Chi, "Vector-valued graph trend filtering with non-convex penalties," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 6, pp. 48–62, Dec. 06 2019, doi: 10.1109/TSIPN.2019.2957717

[16] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1-4, pp. 259–268, Nov. 1992.

[17] A. Chambolle, "An algorithm for total variation minimization and applications," *J. Math. Imag. Vis.*, vol. 20, no. 1/2, pp. 89–97, 2004.

[18] F. R. K. Chung, *Spectral Graph Theory CBMS Regional Conference Series in Mathematics*, vol. 92, 1997.

[19] S. Chen, A. Sandryhaila, J. M. F. Moura, and J. Kovačević, "Signal recovery on graphs: Variation minimization," *IEEE Trans. Signal Process.*, vol. 63, no. 17, pp. 4609–4624, Sep. 2015.

[20] R. Shafipour, A. Khodabakhsh, and G. Mateos, "A windowed digraph fourier transform," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Brighton, U.K., 12–17, 2019, pp. 7525–7529.

[21] D. I. Shuman, C. Wiesmeyr, N. Holighaus, and P. Vandergheynst, "Spectrum-adapted tight graph wavelet and vertex-frequency frames," *IEEE Trans. Signal Process.*, vol. 63, no. 16, pp. 4223–4235, Aug. 2015.

[22] Y. C. Eldar and G. Kutyniok, *Compressed Sensing: Theory and Applications*. Cambridge, U. K.: Cambridge Univ. Press, 2012.

[23] Y. C. Eldar, *Sampling Theory: Beyond Bandlimited Systems*. Cambridge, U.K.: Cambridge Univ. Press, 2015.

[24] S. K. Narang and A. Ortega, "Perfect reconstruction two-channel wavelet filter banks for graph structured data," *IEEE Trans. Signal Process.*, vol. 60, no. 6, pp. 2786–2799, Jun. 2012.

[25] A. Sakiyama, K. Watanabe, Y. Tanaka, and A. Ortega, "Two-channel critically sampled graph filter banks with spectral domain sampling," *IEEE Trans. Signal Process.*, vol. 67, no. 6, pp. 1447–1460, Mar. 2019.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[27] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," 2018, *arXiv:1806.01261*.

[28] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. 2nd Int. Conf. Learn. Representations*, Banff, AB, Canada, 2014, pp. 14–16

[29] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Adv. Neural Inf. Process. Syst. 29th: Annu. Conf. Neural Inf. Process. Syst.*, Barcelona, Spain, 2016, pp. 3837–3845.

[30] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. 6th Int. Conf. Learn. Representations*, Vancouver, BC, Canada, 2018.

[31] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. Devon Hjelm, "Deep graph infomax," in *Proc. 7th Int. Conf. Learn. Representations*, New Orleans, LA, USA, May 2019, pp. 6–9.

[32] F. Wu, A. H. Souza Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proc. 36th Int. Conf. Mach. Learn.*, 9–15 Jun. 2019, Long Beach, California, USA, vol. 97, pp. 6861–6871.

[33] H. Gao and S. Ji, "Graph u-nets," in *Proc. 36th Int. Conf. Mach. Learn.*, 9-15 Jun. 2019, Long Beach, California, USA, vol. 97, pp. 2083–2092.

[34] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. 32nd AAAI Conf. Artif. Intell.*, New Orleans, Louisiana, USA, Feb. 2-7, 2018, pp. 4438–4445.

[35] O. Solomon *et al.*, "Deep unfolded robust PCA with application to clutter suppression in ultrasound," *IEEE Trans. Med. Imag.*, vol. 39, no. 4, pp. 1051–1063, Apr. 2020.

[36] G. Wang, G. B. Giannakis, and J. Chen, "Learning ReLU networks on linearly separable data: Algorithm, optimality, and generalization," *IEEE Trans. Signal Process.*, vol. 67, no. 9, pp. 2357–2370, May 2019.

[37] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst. 31st: Annu. Conf. Neural Inf. Process. Syst.*, Montréal, Canada, 2018, pp. 5171–5181.

[38] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Process. Mag.*, vol. 38, no. 2, pp. 18–44, Mar. 2021.

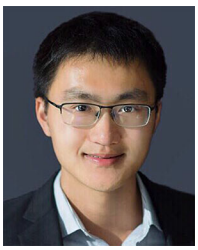
[39] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. 27th Int. Conf. Mach. Learn.*, Haifa, Israel, 2010, pp. 399–406.

[40] L. Zhang, G. Wang, and G. B. Giannakis, "Real-time power system state estimation and forecasting via deep unrolled neural networks," *IEEE Trans. Signal Process.*, vol. 67, no. 15, pp. 4069–4077, Aug. 2019.

[41] Y. Li, M. Tofighi, V. Monga, and Y. C. Eldar, "An algorithm unrolling approach to deep image deblurring," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Brighton, United Kingdom, 12–17, 2019, pp. 7675–7679.

[42] Y. Li, M. Tofighi, J. Geng, V. Monga, and Y. C Eldar, "Efficient and interpretable deep blind image deblurring via algorithm unrolling," *IEEE Trans. Computational Imaging*, vol. 6, pp. 666–681, Jan. 2020.

[43] A. H. Al-Shabili, H. Mansour, and P. T. Boufounos, "Learning plug-and-play proximal quasi-newton denoisers," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2020, pp. 8896–8900.
[44] Y. C. Eldar and G. Dardikman-Yoffe, "Learned SPARCOM: Unfolded deep super-resolution microscopy," *Opt. Exp.*, vol. 28, pp. 27736–27763, 2020.
[45] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013.
[46] S. Chen, R. Varma, A. Singh, and J. Kovačević, "Representations of piecewise smooth signals on graphs," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2016, pp. 6370–6374.
[47] X. Zhu, J. Lafferty, and Z. Ghahramani, "Combining active learning and semi-supervised learning using gaussian fields and harmonic functions," in *Proc. Int. Conf. Mach. Learn. Workshop Continuum Labeled Unlabeled Data Mach. Learn. Data Mining*, 2003, pp. 58–65.
[48] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Proc. Adv. Neural Inf. Process. Syst.*, 2001, pp. 849–856.
[49] S. Segarra, A. G. Marques, and A. Ribeiro, "Linear network operators using node-variant graph filters," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2016, pp. 4850–4854.
[50] M. Coutino, E. Isufi, and G. Leus, "Distributed edge-variant graph filters," in *Proc. IEEE 7th Int. Workshop Comput. Adv. Multi-Sensor Adaptive Process.*, 2017, pp. 1–5.
[51] E. Isufi, F. Gama, and A. Ribeiro, "Edgenets: Edge varying graph neural networks," 2020, *arXiv:2001.07620*.
[52] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 4, pp. 1034–1049, Feb. 2019.
[53] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, New York, NY, USA, 2014, pp. 701–710.
[54] Y. Wang, J. Yang, W. Yin, and Y. Zhang, "A new alternating minimization algorithm for total variation image reconstruction," *SIAM J. Imag. Sci.*, vol. 1, no. 3, pp. 248–272, 2008.
[55] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.
[56] N. Tremblay and P. Borgnat, "Subgraph-based filterbanks for graph signals," *IEEE Trans. Signal Process.*, vol. 64, no. 15, pp. 3827–3840, Aug. 2016.
[57] T. Huu Do, D. Minh Nguyen, and N. Deligiannis, "Graph auto-encoder for graph signal denoising," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2020, pp. 3322–3326.
[58] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in gnns," in *Proc. 8th Int. Conf. Learn. Representations*, Addis Ababa, Ethiopia, Apr. 2020, pp. 26–30.
[59] S. Chen, Y. Yang, S. Zong, A. Singh, and J. Kovačević, "Detecting localized categorical attributes on graphs," *IEEE Trans. Signal Process.*, vol. 65, no. 10, pp. 2725–2740, May 2017.
[60] A. Shocher, S. Bagon, P. Isola, and M. Irani, "Ingan: Capturing and retargeting the "dna" of a natural image," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Seoul, Korea (South), 2019, pp. 4491–4500.
[61] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Deep image prior," *Int. J. Comput. Vis.*, vol. 128, no. 7, pp. 1867–1888, 2020.



**Siheng Chen** (Member, IEEE) received the bachelor's degree in electronics engineering from the Beijing Institute of Technology, Beijing, China, in 2011, and the two master's degrees in electrical and computer engineering and machine learning and the Doctorate degree in 2016 in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA. He is currently an Associate Professor with Shanghai Jiao Tong University, Shanghai, China. Before that, he was a Research Scientist with Mitsubishi Electric Research Laboratories, Cambridge, MA, USA, and an Autonomy Engineer with Uber Advanced Technologies Group, working on the perception and prediction systems of self-driving cars. Before joining Uber, he was a Postdoctoral Research Associate with Carnegie Mellon University. His research interests include graph signal processing, graph neural networks, and autonomous systems. He was the recipient of the 2018 IEEE Signal Processing Society Young Author Best Paper Award and ASME SHM/NDE 2020 Best Journal Paper Award Runner-Up. His coauthored paper was the recipient of the Best Student Paper Award at IEEE GlobalSIP 2018.

**Yonina C. Eldar** (Fellow, IEEE) received the B.Sc. degree in physics and the B.Sc. degree in electrical engineering from Tel-Aviv University, Tel-Aviv, Israel, in 1995 and 19956, respectively, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2002.

She is currently a Professor with the Department of Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel. She was previously a Professor with the Department of Electrical Engineering, Technion, where she held the Edwards Chair in Engineering. She is also a Visiting Professor with MIT, a Visiting Scientist with Broad Institute, and an Adjunct Professor with Duke University, Durham, NC, USA, and was a Visiting Professor with Stanford. She is author of the book *Sampling Theory: Beyond Bandlimited Systems* and co-author of four other books published by Cambridge University Press. Her research interests include statistical signal processing, sampling theory and compressed sensing, learning and optimization methods, and their applications to biology, medical imaging, and optics. She is a Member of the Israel Academy of Sciences and Humanities (elected 2017) and a EURASIP Fellow.

She was the recipient of many awards for excellence in research and teaching, including the IEEE Signal Processing Society Technical Achievement Award (2013), the IEEE/AESS Fred Nathanson Memorial Radar Award (2014), and the IEEE Kiyo Tomiyasu Award (2016). She was a Horev Fellow of the Leaders in Science and Technology Program, Technion and an Alon Fellow. She was the recipient of the Michael Bruno Memorial Award from the Rothschild Foundation, the Weizmann Prize for Exact Sciences, the Wolf Foundation Krill Prize for Excellence in Scientific Research, the Henry Taub Prize for Excellence in Research (twice), the Hershel Rich Innovation Award (three times), the Award for Women with Distinguished Contributions, the Andre and Bella Meyer Lectureship, the Career Development Chair at the Technion, the Muriel David Jacknow Award for Excellence in Teaching, the Technion's Award for Excellence in Teaching (two times), and several best paper awards and best demo awards together with her research students and colleagues, including the SIAM outstanding Paper Prize, the UFFC Outstanding Paper Award, the Signal Processing Society Best Paper Award and the IET Circuits, Devices and Systems Premium Award, was selected as one of the 50 most influential women in Israel and in Asia, and is a highly cited Researcher.

She was a Member of the Young Israel Academy of Science and Humanities and the Israel Committee for Higher Education. She is the Editor-in-Chief of the Foundations and Trends in Signal Processing, a Member of the IEEE Sensor Array and Multichannel Technical Committee and is on several other IEEE committees. In the past, she was a Signal Processing Society Distinguished Lecturer, a Member of the IEEE Signal Processing Theory and Methods and Bio Imaging Signal Processing technical committees, and was an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING, the *EURASIP Journal of Signal Processing*, the *SIAM Journal on Matrix Analysis and Applications*, and the *SIAM Journal on Imaging Sciences*. She was the Co-Chair and Technical Co-Chair of several international conferences and workshops.

**Lingxiao Zhao** received the B.E. degree in electrical engineering from Xi'an Jiaotong University, Xi'an, China, in 2016, and the M.S. degree in 2018 in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, where he is currently working toward the Ph.D. degree in information system with Heinz College. His research interests include deep learning on graphs and many applications with graph structured data.