INVITED PAPER

Graph Neural Networks: Architectures, Stability, and Transferability

This article deals with graph neural networks (GNNs) that operate on data supported on graphs.

By LUANA RUIZ, FERNANDO GAMA, *Member IEEE*, AND ALEJANDRO RIBEIRO, *Member IEEE*

ABSTRACT | Graph neural networks (GNNs) are information processing architectures for signals supported on graphs. They are presented here as generalizations of convolutional neural networks (CNNs) in which individual layers contain banks of graph convolutional filters instead of banks of classical convolutional filters. Otherwise, GNNs operate as CNNs. Filters are composed of pointwise nonlinearities and stacked in layers. It is shown that GNN architectures exhibit equivariance to permutation and stability to graph deformations. These properties help explain the good performance of GNNs that can be observed empirically. It is also shown that if graphs converge to a limit object, a graphon, GNNs converge to a corresponding limit object, a graphon neural network. This convergence justifies the transferability of GNNs across networks with different numbers of nodes. Concepts are illustrated by the application of GNNs to recommendation systems, decentralized collaborative control, and wireless communication networks.

KEYWORDS | Equivariance; graph filters; graph neural networks (GNNs); graph signal processing (GSP); graphon neural networks; graphons; stability; transferability.

I. INTRODUCTION

Graphs can represent lexical relationships in text analysis [1]–[3], product or customer similarities in recommendation systems [4]–[6], or agent interactions in multiagent robotics [7]–[9]. Although otherwise unrelated, these applications share the presence of signals associated with nodes—words, ratings, or perception—out of which we want to extract some information—text categories, ratings of other products, or control actions. If data are available, we can formulate empirical risk minimization (ERM) problems to learn these data-to-information maps. However, it is a form of ERM in which the graph plays a central role in describing relationships between signal components and, therefore, one in which it should be leveraged. Graph neural networks (GNNs) are parameterizations of learning problems, in general, and ERM problems, in particular, that achieve this goal.

In an ERM problem, we are given input–output pairs in a training set, and we want to find a function that best approximates the input–output map according to a given risk (see Section II). This function is later used to estimate the outputs associated with inputs that were not part of the training set. We say that the function has been *trained* and that we have *learned* to estimate outputs. This simple statement hides the fact that the ERM problems do not make sense unless we make assumptions on how the function *generalizes* from the training set to unobserved samples (see Section II-A). We can, for instance, assume that the map is linear, or to be in tune with the times that the map is a deep neural network [10].

A characteristic shared by arbitrary linear and fully connected neural network (FCNN) parameterizations is that they do not scale well with the dimensionality of the input signals. This is best known in the case of signals in Euclidean space—time and images—where many successful examples of scalable linear processing are based on *convolutional* filters and of scalable nonlinear processing on *convolutional* neural networks (CNNs). In this article, we describe *graph* filters [11], [12] and *graph* neural networks [3], [13]–[16] as analogous of convolutional filters and CNNs, but adapted to process signals supported on graphs (see Section III). A graph filter is a

Manuscript received August 1, 2020; revised October 17, 2020 and January 20, 2021; accepted January 26, 2021. Date of publication February 17, 2021; date of current version April 30, 2021. This work was supported by the NSF HDR TRIPODS under Award #1934960. *(Corresponding author: Alejandro Ribeiro.)*

Luana Ruiz and **Alejandro Ribeiro** are with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: rubruiz@seas.upenn.edu; aribeiro@seas.upenn.edu).

Fernando Gama is with the Electrical Engineering and Computer Sciences Department, University of California at Berkeley, Berkeley, CA 94720-5800 USA (e-mail: fgama@berkeley.edu).

Digital Object Identifier 10.1109/JPROC.2021.3055400

polynomial in a matrix representation of the graph. Out of this definition, we build a graph perceptron with the addition of a pointwise nonlinear function to process the output of a graph filter (see Section III-A). Graph perceptrons can be layered to build a multilayer GNN (see Section III-B), and individual layers are augmented from single filters to filter banks to build multiple-feature GNNs (see Section III-C).

At this juncture, an important question is whether graph filters and GNNs do for signals supported on graphs what convolutional filters and CNNs do for Euclidean data. In other words, do they enable scalable processing of signals supported on graphs? A growing body of empirical work shows that this is true to some extent although results are not as impressive as in the case of voice and image processing. As an example that we can use to illustrate the advantages of graph filters and GNNs, consider a recommendation system (see Section II-B) in which we want to use past ratings that customers have given to products to predict future ratings [17]. Collaborative filtering solutions build a graph of product similarities and interpret customer ratings as signals supported on the product similarity graph [4]. We then use past ratings to construct a training set and learn to fill in the ratings that a given customer would give to products not yet rated. Empirical results do show that graph filters and GNNs work in recommendation systems with a large number of products in which linear maps and FCNNs do not [4]–[6]. In fact, this example leads to three empirical observations that motivate this article (see Section III-D).

(O1): Graph filters produce better rating estimates than arbitrary linear parameterizations, and GNNs produce better estimates than arbitrary (fully connected) neural networks, provided that sufficient training data is available.

(O2): GNNs predict ratings better than graph filters.

(O3): A GNN that is trained on a graph with a certain number of nodes can be executed in a graph with a larger number of nodes and still produces good rating estimates.

Observations (O1)–(O3) support advocacy for the use of GNNs, at least in recommendation systems. However, they also spark three interesting questions: (Q1) why do graph filters and GNNs outperform linear transformations and FCNNs? (Q2) why do GNNs outperform graph filters? and (Q3) why do GNNs transfer to networks with a different number of nodes? In this article, we present three theoretical analyses that help answer these questions.

1) *Equivariance:* Graph filters and GNNs are equivariant to permutations of the graph (see Section III).
2) *Stability:* GNNs provide a better tradeoff between discriminability and stability to graph perturbations (see Section IV).
3) *Transferability:* As graphs converge to a limit object, a graphon, GNN outputs converge to outputs of a corresponding limit object, a graphon neural network (see Section V).

These properties show that GNNs have strong *generalization* potential. Equivariance to permutations implies that nodes with analogous neighbor sets making analogous observations perform the same operations. Thus, we can learn to, say, fill in the ratings of a product from the ratings of another product in another part of the network if the local structures of the graph are the same (see Fig. 2). This helps explain why graph filters outperform linear transforms and GNNs outperform FCNNs [see observation (O1)]. Stability to graph deformations affords a stronger version of this statement. We can learn to generalize across different products if the local neighborhood structures are similar, not necessarily identical (see Fig. 3). GNNs possess better stability than graph filters for the same level of discriminability, which helps explain why GNNs outperform graph filters [see observation (O2)]. The convergence of GNNs toward graphon neural networks delineated under the transferability heading explains why GNNs can be trained and executed in graphs of different sizes [see observation (O3)]. It is important to note that analogous to these properties hold for CNNs. They are equivariant to translations and stable to the Euclidean space deformations [18] and have well-defined continuous-time limits.

We focus on a tutorial introduction to GNNs and on describing some of their fundamental properties. This focus renders several relevant questions out of scope. Most notably, we do not discuss training [19], [20]. The role of proper optimization techniques, the selection of proper optimization objectives, and the realization of graph filters are critical in ensuring that the *potential* for generalization implied by equivariance, stability, and transferability is actually *realized.* References for the interested reader are provided in Section I-A.

A. Context and Further Reading

The field of graph signal processing (GSP) has been developed over the last decade [11], [21], [22]. Central to developments in GSP is the notion of graph convolutional filters [11], [12], [21], [23], [24]. GNNs arose as nonlinear extensions of graph filters, obtained by the addition of pointwise nonlinearities to the processing pipeline [3], [13]–[15], [25]. Several implementations of GNNs have been proposed. These include graph convolutional filters implemented in the spectral domain [13], implementations of graph filters with Chebyshev polynomials [3], and ordinary polynomials [14], [26]. One can also encounter GNNs described in terms of local aggregation functions [15], [27]. These can be seen as particular cases of GNNs that use graph filters of order 1 because local aggregation operations can be described as matrix multiplications with some matrix representation of the graph. This results in a parameterization with lower representation power than those in [3], [13], and [14].

It is important to point out that the GNNs in [3], [13], [14] are equivalent in the sense that they span the exact same set of maps. Thus, although we use the polynomial description of [14], the results that we present apply irrespective of implementation. The architectures in [15] and [27], being restricted to filters of order 1, span a subset of the maps that can be represented by the more generic GNNs in [3], [13], and [14]. Hence, results also apply to [15] and [27], except for discriminability discussions that require the use of higher order graph filters. Equivalence notwithstanding, these architectures may differ in their ease of training, leading to a different performance in practice.

GNNs using linear transforms other than graph filters have also been proposed [16], [28]–[30]. Extension of nonlinearities to encompass neighborhood information is proposed in [29], and architectures with residual connections are proposed in [31] and [32]. Edge-varying filters [33] can be used to design edge-varying GNNs [16] and graph attention networks [28], [34]. Multihop attention-based GNNs are introduced in [35]. Architectures considering multirelational data, that is, data with support on multiple graphs or graphs with multidimensional edge features, have been proposed in [31] and [36], and architectures that leverage time dependencies are available in the form of graph recurrent neural networks [30], [37], [38]. We point out that these architectures are different from the GNNs based on graph filters that are described in this article. To stress this point, GNNs that rely on graph convolutional filters are sometimes called graph convolutional neural networks (GCNNs).

Results on permutation equivariance and stability that we present here are drawn from [39] and results on transferability are drawn from [40]. Other important works on the stability of GNNs appear in the context of graph scattering transforms [41], [42]. Permutation equivariance is simple to prove but has, nevertheless, drawn considerable attention because of its practical importance [27], [41]–[44]. Our transferability analysis builds upon the concept of graphons and convergent graph sequences [45], [46] that have proved insightful when processing graph data [47]–[49]. In particular, GSP in the limit has given rise to the topic of graphon signal processing [40], [50], [51]. An alternative transferability analysis relying on generic topological spaces, such as manifolds, where graph Laplacians are sampled from Laplace–Beltrami operators, is also possible [52].

Throughout this article, we use recommendation systems as a running example to illustrate ideas [4]–[6] and, in Sections VI–VII, present numerical results that illustrate GNN stability in decentralized robot control and GNN transferability in wireless resource allocation. The first two are examples of supervised learning problems, whereas the latter is an example of unsupervised learning. These are only some of the problems to which GNNs have been applied successfully; others include identifying brain disorders [53], learning molecule fingerprints [54], web page ranking [55], text categorization [3], [14], and clustering of citation networks [15], [28], [56]. Of particular interest to the electrical engineering community are applications to cyber–physical systems, such as power grids [57], decentralized collaborative control of multiagent robotic systems [7], [9], and wireless communication networks [58].

II. MACHINE LEARNING ON GRAPHS

Consider a graph $\mathbf{G}$ composed of vertices $V = \{1, \ldots, n\}$, edges E defined as ordered pairs (i, j), and weights w_{ij} associated with the edges. Our interest in this article is on machine learning problems defined over this graph. Namely, we are given pairs $(\mathbf{x}, \mathbf{y})$ composed of an input graph signal $\mathbf{x} \in \mathbb{R}^n$ and a target output graph signal $\mathbf{y} \in \mathbb{R}^n$. That $\mathbf{x}$ and $\mathbf{y}$ are graph signals means that the components x_i and y_i are associated with the ith node of the graph. The pair $(\mathbf{x}, \mathbf{y})$ is jointly drawn from a probability distribution $p(\mathbf{x}, \mathbf{y})$, and our goal is to find a function $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ such that $\Phi(\mathbf{x})$ approximates $\mathbf{y}$ over the probability distribution $p(\mathbf{x}, \mathbf{y})$. To do so, we introduce the nonnegative loss function $\ell(\Phi(\mathbf{x}), \mathbf{y}) \geq 0$ such that $\ell(\Phi(\mathbf{x}), \mathbf{y}) = 0$ when $\Phi(\mathbf{x}) = \mathbf{y}$ in order to measure the dissimilarity between the output $\Phi(\mathbf{x})$ and the target output $\mathbf{y}$. We can now define the function $\Phi^\dagger$ that best approximates $\mathbf{y}$ as the one that minimizes the loss $\ell(\Phi(\mathbf{x}), \mathbf{y})$ averaged over the probability distribution $p(\mathbf{x}, \mathbf{y})$

$$\Phi^\dagger = \underset{\Phi}{\operatorname{argmin}}\, \mathbb{E}[\ell(\Phi(\mathbf{x}), \mathbf{y})] = \underset{\Phi}{\operatorname{argmin}} \int \ell(\Phi(\mathbf{x}), \mathbf{y})\, dp(\mathbf{x}, \mathbf{y}). \tag{1}$$

The expectation $\mathbb{E}[\ell(\Phi(\mathbf{x}), \mathbf{y})]$ is said to be a statistical loss, and (1) is termed a statistical loss minimization problem.

A critical condition to solve (1) is availability of the probability distribution $p(\mathbf{x}, \mathbf{y})$. If this is known, the solution to (1) is to compute a posterior distribution that depends on the form of the loss function $\ell(\Phi(\mathbf{x}), \mathbf{y})$. The whole idea of machine learning, though, is that $p(\mathbf{x}, \mathbf{y})$ is not known. Instead, we have access to a collection of Q data samples $(\mathbf{x}_q, \mathbf{y}_q)$ drawn from the distribution $p(\mathbf{x}, \mathbf{y})$ which we group in the training set $\mathcal{T} := \{(\mathbf{x}_q, \mathbf{y}_q)\}_{q=1}^{Q}$. Assuming that these samples are acquired independently and that the number of samples Q is large, a good approximation to the statistical loss in (1) is the empirical average $\bar{\ell}(\Phi) := (1/Q)\sum_{q=1}^{Q} \ell(\Phi(\mathbf{x}_q), \mathbf{y}_q)$. Therefore, it is sensible to change our objective to search for a function Φ^* that minimizes the empirical average $\bar{\ell}(\Phi)$

$$\Phi^* = \underset{\Phi}{\operatorname{argmin}}\, \frac{1}{Q} \sum_{q=1}^{Q} \ell(\Phi(\mathbf{x}_q), \mathbf{y}_q). \tag{2}$$

We say that (2) is an ERM problem. The function Φ^* is the optimal *empirical* function associated with the training set $\mathcal{T}$.

A. Learning Parameterizations

Observe that the solution to (2) is elementary. Since $\ell(\Phi(\mathbf{x}), \mathbf{y}) = 0$ when $\Phi(\mathbf{x}) = \mathbf{y}$ and nonnegative otherwise,

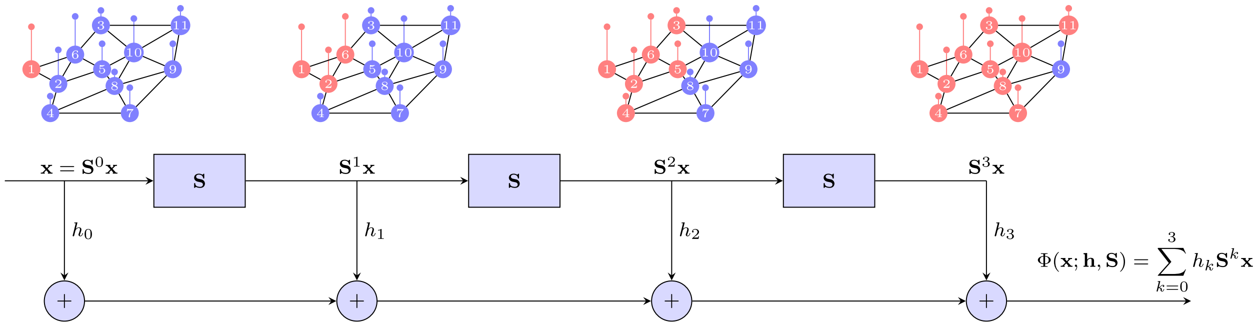


Fig. 1. ***Graph convolutional filter is a polynomial on a matrix representation of the graph S. We think of them as operations that propagate information through adjacent nodes. As the order of the filter grows, we aggregate information from nodes that are farther apart. However, the integration of this information is always mediated by the neighborhood structure of the graph.***

it suffices to make $\Phi(\mathbf{x}_q) = \mathbf{y}_q$ for all the observed samples $\mathbf{x}_q$—or some sort of average if the same input $\mathbf{x}_q$ is observed several times. However, (2) only makes sense as a problem formulation if we have access to all possible samples $\mathbf{x}_q$. However, the interest in practice is to infer, or *to learn*, the value of $\mathbf{y}$ for samples $\mathbf{x}$ that have not been observed before.

This motivates the introduction of a learning parameterization $\mathcal{H}$ that restricts the family of functions Φ that are admissible in (2). Thus, instead of searching over all $\Phi(\mathbf{x})$'s, we search over functions $\Phi(\mathbf{x}; \mathcal{H})$ so that the ERM problem in (2) is replaced by the alternative ERM formulation

$$\mathcal{H}^* = \underset{\mathcal{H}}{\operatorname{argmin}} \frac{1}{Q} \sum_{q=1}^{Q} \ell(\Phi(\mathbf{x}_q; \mathcal{H}), \mathbf{y}_q). \tag{3}$$

A particular choice of parameterization is the set of linear functions of the form $\Phi(\mathbf{x}; \mathbf{H}) = \mathbf{H}\mathbf{x}$, in which case (2) becomes

$$\mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \frac{1}{Q} \sum_{q=1}^{Q} \ell(\mathbf{H}\mathbf{x}_q, \mathbf{y}_q). \tag{4}$$

Alternatively, one could choose $\Phi(\mathbf{x}; \mathcal{H})$ to be a neural network, or as we will advocate in Section III, a graph filter or a GNN. The important point to highlight here is that the design of a machine learning system is tantamount to the selection of the proper learning parameterization. This is because in (3), the only choice left for a system designer is the class of functions $\Phi(\mathbf{x}; \mathcal{H})$ spanned by different choices of $\mathcal{H}$. However, more importantly, this is also because the choice of parameterization determines how the function $\Phi(\mathbf{x}; \mathcal{H})$ generalizes from (observed) samples in the training set $(\mathbf{x}_q, \mathbf{y}_q) \in \mathcal{T}$ to unobserved signals $\mathbf{x}$.

B. Recommendation Systems

An example of an ERM problem involving graph signals is a collaborative filtering approach to recommendation systems [4]. In a recommendation system, we want to predict the ratings that customers would give to a certain product using rating histories. Collaborative filtering solutions build a graph of product similarities using past ratings and look at the ratings of each customer as a graph signal supported on the nodes of the product graph.

1) Product Similarity Graph: Denote by x_{ci} the rating that customer c gives to product i. Typically, product i has been rated by a subset of customers that we denote $\mathcal{C}_i$. We consider the sets of users $\mathcal{C}_{ij} = \mathcal{C}_i \cap \mathcal{C}_j$ that have rated products i and j and compute correlations

$$\sigma_{ij} = \frac{1}{|\mathcal{C}_{ij}|} \sum_{c \in \mathcal{C}_{ij}} (x_{ci} - \mu_{ij})(x_{cj} - \mu_{ji}) \tag{5}$$

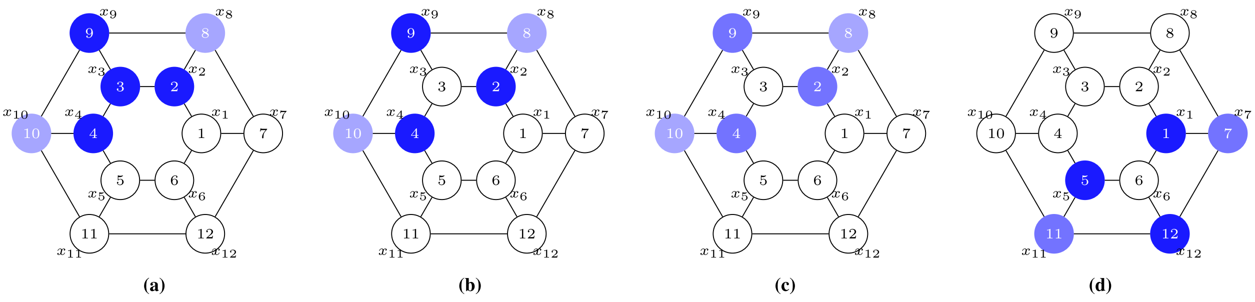
where we use the average ratings $\mu_{ij} = (1/|\mathcal{C}_{ij}|) \sum_{c \in \mathcal{C}_{ij}} x_{ci}$ and $\mu_{ji} = (1/|\mathcal{C}_{ij}|) \sum_{c \in \mathcal{C}_{ij}} x_{cj}$. The product graph used in collaborative filtering is the one with normalized weights

$$w_{ij} = \sigma_{ij} \Big/ \sqrt{\sigma_{ii}\sigma_{jj}}. \tag{6}$$

A cartoon illustration of the product graph is shown in Fig. 2(a). Nodes represent different products, edges stand in for product similarity, and signal components are the product ratings of a given customer. As is typical in practice, a small number of products have been rated.

2) Training Set: To build a training set for this problem, define the vector $\mathbf{x}_c = [x_{c1}; \ldots; x_{cn}]$, where x_{ci} is the rating that user c gave to product i if available or $x_{ci} = 0$ otherwise. Further denote as $\mathcal{I}_c$ the set of items rated by customer c. Let $i \in \mathcal{I}_c$ be a product rated by customer c and define the sparse vector $\mathbf{y}_{ci}$ whose unique nonzero entry is $[\mathbf{y}_{ci}]_i = x_{ci}$. With these definitions, we construct the training set

$$\mathcal{T} = \bigcup_{c, i \in \mathcal{I}_c} \{(\mathbf{x}_{ci}, \mathbf{y}_{ci}) \, : \, \mathbf{x}_{ci} = \mathbf{x}_c - \mathbf{y}_{ci}\}. \tag{7}$$



(a) (b) (c) (d)

Fig. 2. ***Graph represents product similarity in a recommendation system. If we are given samples (a) for training, any reasonable parameterization learns to complete the rating of node 3 when observing the signal in (b). The linear parameterization in (4) also learns to fill the rating of node 3 when observing (c)—node saturation is proportional to the signal value. The graph filter parameterization in (13) generalizes to (c), but it also generalizes to predicting the rating of node 6 in (d). This is true because of the permutation equivariance result in Proposition 1. GNNs [see (21)–(23)] inherit this generalization property (see Proposition 2).***

The process of building an input–output pair of the training set is illustrated in Fig. 2(b). In this particular example, we isolate the rating that this customer gave to product $i = 3$. This rating is recorded into a graph signal with a single nonzero entry $[\mathbf{y}_{c3}]_3 = x_{c3}$. The remaining nonzero entries define the rating input $\mathbf{x}_{c3} = \mathbf{x}_c - \mathbf{y}_{c3}$. This process is repeated for all the products in the set $i \in \mathcal{I}_c$ of rated items of costumer c and for all customers c.

3) Loss Function: Our goal is to learn a map that will produce outputs $\mathbf{y}_{ci}$ when presented with inputs $\mathbf{x}_{ci}$. For example, in the case of Fig. 2, we want to present Fig. 2(b) as an input and fill in a rating of product $i = 3$ equal to the rating of product $i = 3$ in Fig. 2(a). To do that, we define the loss function

$$\ell(\Phi(\mathbf{x}_{ci}; \mathcal{H}), \mathbf{y}_{ci}) = \frac{1}{2}\left(\mathbf{e}_i^T \Phi(\mathbf{x}_{ci}; \mathcal{H}) - \mathbf{e}_i^T \mathbf{y}_{ci}\right)^2 \tag{8}$$

where the vector $\mathbf{e}_i$ is the ith entry of the canonical basis of $\mathbb{R}^n$. Since multiplying with $\mathbf{e}_i^T$ extracts the ith component of a vector, the loss in (8) compares the predicted rating $\mathbf{e}_i^T \Phi(\mathbf{x}_{ci}; \mathcal{H}) = [\Phi(\mathbf{x}_{ci}; \mathcal{H})]_i$ with the observed rating $\mathbf{e}_i^T \mathbf{y}_{ci} = [\mathbf{y}_{ci}]_i = x_{ci}$. At execution time, this map can be used to predict ratings of unrated products from the ratings of rated products. If we encounter the signal in Fig. 2(b), we know the prediction will be accurate because we encountered this signal during training. If we are given the signals in Fig. 2(c) or (d), successful rating predictions depend on the choice of parameterization.

III. GRAPH NEURAL NETWORKS

As we explained in Section II-A, the choice of parameterization determines the manner in which the function $\Phi(\mathbf{x}; \mathcal{H})$ generalizes from elements of the training set to unobserved samples. A parameterization that is convenient for processing graph signals is a graph convolutional filter [11], [12], [21], [23]. To define this operation, let $\mathbf{S} \in \mathbb{R}^{n \times n}$ denote a matrix representation of the graph and introduce a filter-order K along with filter coefficients h_k that we group in the vector $\mathbf{h} = [h_0; \ldots; h_K]$. A graph convolutional filter applied to the graph signal $\mathbf{x}$ is a polynomial on this matrix representation

$$\mathbf{u} = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x} = \Phi(\mathbf{x}; \mathbf{h}, \mathbf{S}) \tag{9}$$

where we have defined $\Phi(\mathbf{x}; \mathbf{h}, \mathbf{S})$ in the second equality to represent the output of a graph filter with coefficients $\mathbf{h}$ run on the matrix representation $\mathbf{S}$ and applied to the graph signal $\mathbf{x}$. The output $\mathbf{u} = \Phi(\mathbf{x}; \mathbf{h}, \mathbf{S})$ is also a graph signal. In the context of (9), the representation $\mathbf{S}$ is termed a graph shift operator. If we need to fix ideas, we will interpret $\mathbf{S}$ as the adjacency matrix of the graph with entries $S_{ij} = w_{ij}$, but nothing really changes if instead we work with the Laplacian or normalized versions of the adjacency or Laplacian [22].

One advantage of graph filters is their locality. Indeed, we can define the diffusion sequence as the collection of graph signals $\mathbf{z}_k = \mathbf{S}^k \mathbf{x}$ to rewrite the filter in (9) as $\mathbf{u} = \sum_{k=0}^{K} h_k \mathbf{z}_k$. It is ready to see that the diffusion sequence is given by the recursion $\mathbf{z}_k = \mathbf{S}\mathbf{z}_{k-1}$ with $\mathbf{z}_0 = \mathbf{x}$. Further observing that $S_{ij} \neq 0$ only when the pair (i, j) is an edge of the graph, we see that the entries of the diffusion sequence satisfy

$$z_{k,i} = \sum_{j:(i,j)\in\mathcal{E}} S_{ij} z_{k-1,j}. \tag{10}$$

We can, therefore, interpret the graph filter in (9) as an operation that propagates information through adjacent nodes, as we illustrate in Fig. 1. This is a property that graph convolutional filters share with regular convolutional filters in time and offers motivation for their use in the processing of graph signals.

In the context of machine learning on graphs, a more important property of graph filters is their *equivariance to permutation*. Use $\mathbf{P}$ to denote a permutation matrix—entries P_{ij} are binary with exactly one nonzero entry in each row and column. The vector $\hat{\mathbf{x}} = \mathbf{P}\mathbf{x}$ is just a reordering of the entries of $\mathbf{x}$ that we can interpret as a graph signal supported on the graph $\hat{\mathbf{S}} = \mathbf{P}\mathbf{S}\mathbf{P}^T$, which is just a reordering of the graph $\mathbf{S}$. When processing of $\hat{\mathbf{x}}$ on the graph $\hat{\mathbf{S}}$ with the graph filter $\mathbf{h}$, the following proposition from [39], originally proved in [11], holds.

Proposition 1: Graph filters are permutation equivariant

$$\Phi(\hat{\mathbf{x}}; \mathbf{h}, \hat{\mathbf{S}}) = \Phi(\mathbf{P}\mathbf{x}; \mathbf{h}, \mathbf{P}\mathbf{S}\mathbf{P}^T) = \mathbf{P}\Phi(\mathbf{x}; \mathbf{h}, \mathbf{S}). \qquad (11)$$

Proof: Use the definitions of the graph filter in (9) and the permutations $\hat{\mathbf{x}} = \mathbf{P}\mathbf{x}$ and $\hat{\mathbf{S}} = \mathbf{P}\mathbf{S}\mathbf{P}^T$ to write

$$\Phi(\hat{\mathbf{x}}; \mathbf{h}, \hat{\mathbf{S}}) = \sum_{k=0}^{K} h_k \hat{\mathbf{S}}^k \hat{\mathbf{x}} = \sum_{k=0}^{K} h_k (\mathbf{P}\mathbf{S}\mathbf{P}^T)^k \mathbf{P}\mathbf{x}. \qquad (12)$$

Since $\mathbf{P}^T\mathbf{P} = \mathbf{I}$ for any permutation matrix, (11) follows. ■

We include the proof of Proposition 1 to highlight that this is an elementary result. Its immediate relevance is that it shows that processing a graph signal with a graph filter is independent of node labeling. This is something we know must hold in several applications—it certainly must hold for the recommendation problem described in Section II-B—but that is not true of, say, the linear parameterization in (4). There is, however, further value in permutation equivariance. To explain this, return to the ERM problem in (3) and utilize the graph filter in (9) as a learning parameterization. This yields the learning problem

$$\mathbf{h}^* = \underset{\mathbf{h}}{\operatorname{argmin}} \frac{1}{Q} \sum_{q=1}^{Q} \ell\left(\sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x}_q, \, \mathbf{y}_q \right). \qquad (13)$$

An important observation is that we know that (4) must yield a function $\Phi(\mathbf{x}; \mathbf{H}^*)$ whose average loss is smaller than the average loss attained by the function $\Phi(\mathbf{x}; \mathbf{h}^*, \mathbf{S})$ obtained from solving (13). This is because both are linear transformations, and while $\Phi(\mathbf{x}; \mathbf{H}) = \mathbf{H}\mathbf{x}$ is generic, the graph filter $\Phi(\mathbf{x}; \mathbf{h}, \mathbf{S}) = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x}$ belongs to a particular linear class. This is certainly true on the training set $\mathcal{T}$, but, when operating on unobserved samples $\mathbf{x}$, the graph filter can and will do better (see results in Section III-D) because its permutation equivariance induces better generalization.

An illustration of this phenomenon is shown in Fig. 2. The graph represents a user similarity network in a recommendation system for which the ratings, as shown in Fig. 2(a), are available at the time of training. Out of these ratings, we can create the graph signal in Fig. 2(b) to add to the training set, and we assume that both parameterizations, the arbitrary linear transformation $\Phi(\mathbf{x}; \mathbf{H}^*)$ in (4) and the graph filter $\Phi(\mathbf{x}; \mathbf{h}^*, \mathbf{S})$ in (13), learn to estimate the rating of user 3 successfully. If this happens, the functions $\Phi(\mathbf{x}; \mathbf{H}^*)$ and $\Phi(\mathbf{x}; \mathbf{h}^*, \mathbf{S})$ also learn to estimate the rating of user 3 when given the signal in Fig. 2(c)—where we interpret colors as proportional to signal values. Notice that this happens even if signals of this form are not observed during training. We say that $\Phi(\mathbf{x}; \mathbf{H}^*)$ and $\Phi(\mathbf{x}; \mathbf{h}^*, \mathbf{S})$ *generalize* to this example.

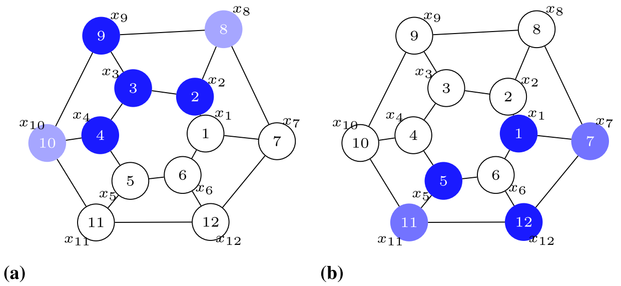


Fig. 3. ***Perfect symmetry as in Fig. 2 is unlikely in practice, but near permutation symmetries can and do appear. We still expect some level of generalization from graph filters [see (13)] and GNNs [see (21)–(23)]. (a) Graph and signal observed at training time. (b) Graph and signal observed at test time.***

If we now consider the signal in Fig. 2(d), the linear parameterization $\Phi(\mathbf{x}; \mathbf{H}^*)$ may or may not generalize to this example. In principle, it would not. The graph filter $\Phi(\mathbf{x}; \mathbf{h}^*, \mathbf{S})$, however, does generalize. This can be seen intuitively from the definition of the diffusion sequence in (10). Whatever operations are done to estimate the rating of user 3 from its adjacent nodes 2, 4, and 9 are the same as those done to estimate the rating of user 6 from its adjacent nodes 1, 5, and 12. More formally, when graphs present symmetries in the sense that they are invariant to some permutation, that is, $\mathbf{S} = \mathbf{P}\mathbf{S}\mathbf{P}^T$, Proposition 1 tells us that $\Phi(\mathbf{P}\mathbf{x}; \mathbf{h}, \mathbf{S}) = \mathbf{P}\Phi(\mathbf{x}; \mathbf{h}, \mathbf{S})$, that is, these operations are equivariant so that the rating prediction is consistent with this relabeling. This is the case of the graph in Fig. 2, which can be permuted onto itself to map the signal in Fig. 2(d) onto the signal in Fig. 2(a). Thus, the graph filter *generalizes* from the example in Fig. 2(a) to fill the rating in Fig. 2(d).

This illustration highlights the generalization properties of graph filters vis-à-vis those of linear transforms. In reality, we are unlikely to encounter the perfect permutation symmetry of Fig. 2. Near permutation symmetry as in Fig. 3 is more expected. In this case, the ability to generalize from Fig. 3(a) to (b) is not as much as the ability to generalize from Fig. 2(a) to (d), but the continuity of (9) dictates that some amount of predictive power extends from observing samples Fig. 3(a) toward the estimation of the rating of user 6 when given the signal in Fig. 3(b).

A. Graph Perceptrons

GNNs extend graph filters by using pointwise nonlinearities that are nonlinear functions that are applied independently to each component of a vector. For a formal definition, begin by introducing a single variable function $\sigma : \mathbb{R} \to \mathbb{R}$ that we extend to the vector function $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ by independent application to each component. Thus,

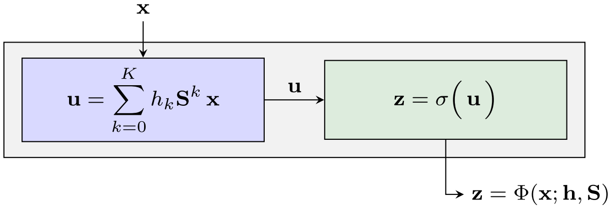

$\mathbf{x}$
$\mathbf{u} = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x}$
$\mathbf{u}$
$\mathbf{z} = \sigma\big(\mathbf{u}\big)$
$\mathbf{z} = \Phi(\mathbf{x}; \mathbf{h}, \mathbf{S})$

Fig. 4. ***Graph perceptron composes a graph convolutional filter with a pointwise nonlinearity. It is a minor variation of a graph filter that, among other shared properties, retains permutation equivariance.***

if we have $\mathbf{u} = [u_1; \ldots; u_n] \in \mathbb{R}^n$, the output vector $\sigma(\mathbf{u})$ is such that

$$\sigma(\mathbf{u}) \; : \; [\sigma(\mathbf{u})]_i = \sigma(u_i). \tag{14}$$

That is, the output vector is of the form $\sigma(\mathbf{u}) = [\sigma(u_1), \ldots, \sigma(u_n)]$. Observe that we are using σ to denote both the scalar function and the pointwise vector function.

In a single-layer GNN, the graph signal $\mathbf{u}$ is passed through a pointwise nonlinear function satisfying (14) to yield

$$\mathbf{z} = \sigma(\mathbf{u}) = \sigma\left(\sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x}\right). \tag{15}$$

We say that the transform in (15) is a graph perceptron (see Fig. 4). Different from the graph filter in (9), the graph perceptron is a nonlinear function of the input. It is, however, a very simple form of nonlinear processing because the nonlinearity does not mix signal components. Signal components are mixed by the graph filter but are then processed element-wise through σ. In particular, (15) retains the locality properties of graph convolutional filters (see Fig. 1) and their permutation equivariance (see Fig. 2 and Proposition 1).

B. Multiple-Layer Networks

Graph perceptrons can be stacked in layers to create multilayer GNNs (see Fig. 5). This stacking is mathematically written as a function composition where the outputs of a layer become inputs to the next layer. For a formal definition, let $l = 1, \ldots, L$ be a layer index and $\mathbf{h}_l = \{h_{lk}\}_{k=0}^{K}$ be collections of $K+1$ graph filter coefficients associated with each layer. Each of these sets of coefficients defines a respective graph filter $\Phi(\mathbf{x}; \mathbf{h}_l, \mathbf{S}) = \sum_{k=0}^{K} h_{lk} \mathbf{S}^k \mathbf{x}$. At layer l, we take as input the output $\mathbf{x}_{l-1}$ of layer $l-1$ that we process with the filter $\Phi(\mathbf{x}; \mathbf{h}_l, \mathbf{S})$ to produce the intermediate feature

$$\mathbf{u}_l = \mathbf{H}_l(\mathbf{S})\,\mathbf{x}_{l-1} = \sum_{k=0}^{K} h_{lk} \mathbf{S}^k \mathbf{x}_{l-1} \tag{16}$$

where, by convention, we say that $\mathbf{x}_0 = \mathbf{x}$ so that the given graph signal $\mathbf{x}$ is the GNN input. As for the graph perceptron, this feature is passed through a pointwise nonlinear function (which is the same in all layers) to produce the lth layer output

$$\mathbf{x}_l = \sigma(\mathbf{u}_l) = \sigma\left(\sum_{k=0}^{K} h_{lk} \mathbf{S}^k \mathbf{x}_{l-1}\right). \tag{17}$$

After recursive repetition of (16) and (17) for $l = 1, \ldots, L$, we reach $\mathbf{x}_L$, which is not further processed and is declared the GNN output $\mathbf{z} = \mathbf{x}_L$. To represent the GNN output, we define the filter matrix $\mathbf{H} := \{\mathbf{h}_l\}_{l=1}^{L}$ grouping the L sets of filter coefficients at each layer and define the operator $\Phi(\,\cdot\,; \mathbf{H}, \mathbf{S})$ as

$$\Phi(\mathbf{x}; \mathbf{H}, \mathbf{S}) = \mathbf{x}_L. \tag{18}$$

We stress that, in (18), the GNN output $\Phi(\mathbf{x}; \mathbf{H}, \mathbf{S}) = \mathbf{x}_L$ follows from recursive application of (16) and (17) for $l = 1, \ldots, L$ with $\mathbf{x}_0 = \mathbf{x}$. This operator notation emphasizes that the output of a GNN depends on the filter $\mathbf{H}$ and the graph shift operator $\mathbf{S}$. A block diagram for a GNN with $L = 3$ layers is shown in Fig. 5.

The sets of filter coefficients $\mathbf{H}$ that define the GNN operator in (18) are chosen to minimize a training loss, as in (3)

$$\mathbf{H}^* = \operatorname*{argmin}_{\mathbf{H}} \frac{1}{Q} \sum_{q=1}^{Q} \ell(\Phi(\mathbf{x}_q; \mathbf{H}, \mathbf{S}), \mathbf{y}_q). \tag{19}$$

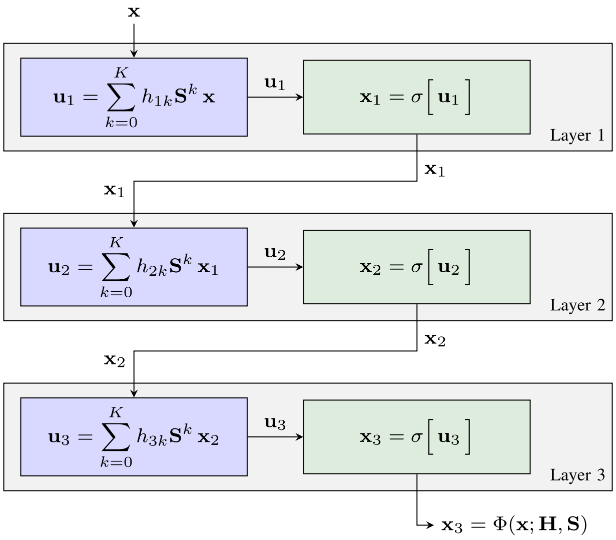

$\mathbf{x}$
$\mathbf{u}_1 = \sum_{k=0}^{K} h_{1k} \mathbf{S}^k \mathbf{x}$
$\mathbf{u}_1$
$\mathbf{x}_1 = \sigma\big[\mathbf{u}_1\big]$
Layer 1
$\mathbf{x}_1$
$\mathbf{x}_1$
$\mathbf{u}_2 = \sum_{k=0}^{K} h_{2k} \mathbf{S}^k \mathbf{x}_1$
$\mathbf{u}_2$
$\mathbf{x}_2 = \sigma\big[\mathbf{u}_2\big]$
Layer 2
$\mathbf{x}_2$
$\mathbf{x}_2$
$\mathbf{u}_3 = \sum_{k=0}^{K} h_{3k} \mathbf{S}^k \mathbf{x}_2$
$\mathbf{u}_3$
$\mathbf{x}_3 = \sigma\big[\mathbf{u}_3\big]$
Layer 3
$\mathbf{x}_3 = \Phi(\mathbf{x}; \mathbf{H}, \mathbf{S})$

Fig. 5. ***GNNs are compositions of layers each of which composes graph filters $\Phi(x; h_l, S) = \sum_{k=0}^{K} h_{lk} S^k$ with pointwise nonlinearities σ [see (16) and (17)]. The output $\Phi(x; H, S) = x_L = x_3$ follows at the end of a cascade of three layers recursively applied to the input x. Layers are defined by sets of coefficients grouped in the matrix $H := \{h_1, h_2, h_3\}$, which is chosen to minimize a training loss for a given shift S [see (3) and (19)].***

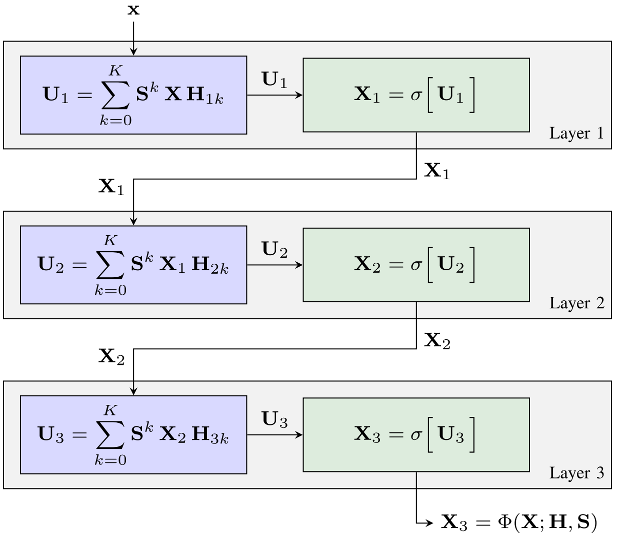

$\mathbf{x}$
$\mathbf{U}_1 = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X} \mathbf{H}_{1k}$
$\mathbf{U}_1$
$\mathbf{X}_1 = \sigma\left[\mathbf{U}_1\right]$
Layer 1
$\mathbf{X}_1$
$\mathbf{X}_1$
$\mathbf{U}_2 = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X}_1 \mathbf{H}_{2k}$
$\mathbf{U}_2$
$\mathbf{X}_2 = \sigma\left[\mathbf{U}_2\right]$
Layer 2
$\mathbf{X}_2$
$\mathbf{X}_2$
$\mathbf{U}_3 = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X}_2 \mathbf{H}_{3k}$
$\mathbf{U}_3$
$\mathbf{X}_3 = \sigma\left[\mathbf{U}_3\right]$
Layer 3
$\mathbf{X}_3 = \Phi(\mathbf{X}; \mathbf{H}, \mathbf{S})$

Fig. 6. ***We expand the representation power of GNNs with the addition of multiple features per layer [see (20)]. The graph filters in each layer are MIMO graph filters [see (21)]. They take F_{l-1} graph signals as inputs and produce F_l graph signals as outputs. The structure is otherwise identical to the single-feature GNN in Fig. 5.***

We emphasize that, similar to the case of the graph filters in (13), the optimization is over the filter matrix $\mathbf{H}$ with the shift operator $\mathbf{S}$ given. We also note that, since each perceptron is permutation equivariant, the whole GNN also inherits the permutation equivariance of graph filters.

C. Multiple-Feature Networks

To further increase the representation power of GNNs, we incorporate multiple features per layer that are the result of processing multiple input features with a bank of graph filters (see Fig. 6). For a formal definition, let F_l be the number of features at layer l and define the feature matrix as

$$\mathbf{X}_l = \left[\mathbf{x}_l^1, \mathbf{x}_l^2, \ldots, \mathbf{x}_l^{F_l}\right]. \tag{20}$$

We have that $\mathbf{X}_l \in \mathbb{R}^{n \times F_l}$ and interpret each column of $\mathbf{X}_l$ as a graph signal. The outputs of Layer $l-1$ are inputs to Layer l where the set of F_{l-1} features in $\mathbf{X}_{l-1}$ are processed by a filterbank made up of $F_{l-1} \times F_l$ filters. For a compact representation of this bank, consider coefficient matrices $\mathbf{H}_{lk} \in \mathbb{R}^{F_{l-1} \times F_l}$ to build the intermediate feature matrix

$$\mathbf{U}_l = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X}_l \, \mathbf{H}_{lk}. \tag{21}$$

Each of the F_l columns of the matrix $\mathbf{U}_l \in \mathbb{R}^{n \times F_l}$ is a separate graph signal. We say that (21) represents a multiple-input–multiple-output (MIMO) graph filter since it takes F_{l-1} graph signals as inputs and yields F_l graph signals at its output. As in the case of the single-feature GNN of Section III-B—and the graph perceptron in (15)—the intermediate feature $\mathbf{U}_l$ is passed through a pointwise nonlinearity to produce the lth layer output

$$\mathbf{X}_l = \sigma(\mathbf{U}_l) = \sigma\left(\sum_{k=0}^{K} \mathbf{S}^k \mathbf{X}_{l-1} \mathbf{H}_{lk}\right). \tag{22}$$

When $l = 0$, we convene that $\mathbf{X}_0 = \mathbf{X}$ is the input to the GNN, which is made of F_0 graph signals. The output $\mathbf{X}_L$ of layer L is also the output of the GNN, which is made up of F_L graph signals. To define a GNN operator, we group filter coefficients $\mathbf{H}_{lk}$ in the tensor $\mathbf{H} = \{\mathbf{H}_{lk}\}_{l,k}$ and define the GNN operator

$$\Phi(\mathbf{X}; \mathbf{H}, \mathbf{S}) = \mathbf{X}_L. \tag{23}$$

If the input is a single graph signal, as in (15) and (18), we have $F_0 = 1$ and $\mathbf{X}_0 = \mathbf{x} \in \mathbb{R}^n$. If the output is also a single graph signal—as is also the case in (15) and (18)—we have $F_L = 1$ and $\mathbf{X}_L = \mathbf{x}_L \in \mathbb{R}^n$.

The sets of filter coefficients $\mathbf{H}$ that define the multiple-feature GNN operator in (23) are chosen to minimize a training loss

$$\mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \frac{1}{Q} \sum_{q=1}^{Q} \ell(\Phi(\mathbf{X}_q; \mathbf{H}, \mathbf{S}), \mathbf{Y}_q) \tag{24}$$

which differs from (19) in that inputs, outputs, and intermediate layers may be composed of multiple features. Each layer of the GNN is made up of filter banks that are permutation equivariant. Since pointwise nonlinearities do not mix signal components, each individual layer is permutation equivariant. It follows that the GNN, being a composition of permutation equivariant operators, is also permutation equivariant. This is a sufficiently important fact that deserves to be highlighted as a proposition that we take from [39].

Proposition 2: GNNs are permutation equivariant

$$\Phi(\hat{\mathbf{x}}; \mathbf{H}, \hat{\mathbf{S}}) = \Phi(\mathbf{P}\mathbf{x}; \mathbf{H}, \mathbf{P}\mathbf{S}\mathbf{P}^T) = \mathbf{P}\Phi(\mathbf{x}; \mathbf{H}, \mathbf{S}). \tag{25}$$

Proposition 2 entails the same comments that follow Proposition 1. In particular, GNNs are expected to generalize from observing the signal in Fig. 2(a) to successfully fill in ratings when presented with the signal in Fig. 2(d), even if this signal is never observed during training. This is an attribute that is not expected of FCNNs and that we verify experimentally in Section III-D. Likewise, we expect generalization to also hold in the case of Fig. 3. As we will see in Section IV, the fundamental difference between GNNs and graph filters is the ability of the former to provide better generalization when signals are close to permutation equivariant but not exactly so.

Remark 1: As is the case of the single-feature filter in (9), we can write the MIMO graph filter in (21) in terms of a diffusion sequence. To do that, we define $\mathbf{Z}_{lk} := \mathbf{S}^k\mathbf{X}_l$ and observe that we can rewrite the matrices $\mathbf{Z}_{lk}$ in the recursive form

$$\mathbf{Z}_{lk} = \mathbf{S}\mathbf{Z}_{l,k-1}, \quad \text{with } \mathbf{Z}_{l0} = \mathbf{X}_l. \tag{26}$$

With this definition, the graph filter in (21) is rewritten as

$$\mathbf{U}_l = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X}_l\, \mathbf{H}_{lk}, = \sum_{k=0}^{K} \mathbf{Z}_{lk}\, \mathbf{H}_{lk}. \tag{27}$$

The use of the diffusion sequence in (27) highlights that the MIMO graph filter accepts a local implementation [see (10)]. This is important in, for example, the use of GNNs in decentralized collaborative systems (see Section VI).

Remark 2: To keep the representation dimension under control, many architectures implement pooling as an intermediate step between the convolutional filter banks and the nonlinearity. Pooling is a summarizing operation that reduces dimensionality by first computing local summaries of the signal and then subsampling it. Permutation equivariance is preserved if the subsampling operation is based on topological features of the graph, such as the node degrees [14]. Pooling strategies for GNNs have been discussed in [3], [13], [14], and [59].

D. Recommendation System Experiments

To illustrate the problem of recommendation systems with a specific numerical example, we consider the MovieLens-100k data set [17] that consists of 100 000 ratings given by 943 users to 1682 movies. These ratings are integers between 1 and 5, and nonexisting ratings are set to 0. The movie similarity network is built by computing similarity scores between pairs of movies, as described in Section II-B. On this network, each user's rating vector $\mathbf{x}_c$ can be represented as a graph signal.

1) Different Parameterizations: In the first experiment, the goal is to predict the ratings to the six movies with most ratings in the data set by solving the ERM problem in (3) with different parameterizations of Φ. In order to do this, we follow the methodology in Section II-B to obtain 3044 input–output pairs corresponding to users who have rated these movies. These data are then split into two: 90% for training (of which 10% are used for validation) and 10% for testing.

Seven different parameterizations were considered: a simple linear parameterization, a graph filter (9), an FCNN, a graph perceptron (15), a multilayer GNN (17), and a single-layer and a multilayer multi-feature GNNs (22). Their hyperparameters are presented in Table 1. Note that the graph filter and GNNs have a readout layer mapping F_L features per node to a single output feature per node, adding F_L extra parameters. All architectures were trained simultaneously by optimizing the L1 loss on the training set, using ADAM with a learning rate of 5×10^{-3} and decay factors of 0.9 and 0.999. The number of epochs and batch size was 40 and 5, respectively.

Table 1 Hyperparameters and Total Number of Parameters of Seven Parameterizations of Φ in (3). The Number of Features, Filter Taps, and Hidden Units Are Denoted *F*, *K*, and *N*, Respectively. For Multilayer Models, F_l/N_l Indicates the Value of These Hyperparameters at Layer *l*

Architecture	L	Hyperparameters	Params.	σ
Linear	-	$n \times n$ matrix	2.8E+6	-
Graph filter	-	$F = 64,\ K = 5$	384	-
FCNN	2	$N_1 = 64,\ N_2 = 32$	1.6E+5	ReLU
Graph perceptron	1	$K = 5$	6	ReLU
GNN	2	$F = 1,\ K = 5$	11	ReLU
GNN	1	$F = 64,\ K = 5$	384	ReLU
GNN	2	$F_1 = 64,\ F_2 = 32,\ K = 5$	1E+4	ReLU

In Table 2, we report the average root mean square error (RMSE) achieved by each parameterization for ten data splits. We observe that the graph filter achieves a much smaller error than the generic linear parameterization while having significantly fewer parameters, which is empirical evidence of its superior ability to exploit the structure of graph signals through permutation equivariance, as discussed in Section III. The fact that the average RMSE of the FCNN, which also has in the order of 10^5 parameters, is worse than those of the GNNs, graph perceptrons, and graph filter can be explained by the same reason, even if the FCNN improves upon the linear transformation due to the nonlinearities. The graph perceptron and the multilayer GNN are not better than the graph filter and showcase similar RMSEs. On the other hand, the addition of multiple features in the single-layer and multilayer GNNs provides sensible improvements, with the two-layer GNN performing better than all other architectures. It turns out that nonlinearities also play an important role in GNN performance, which we examine in the stability discussion of Section IV.

2) GNN Transferability: In the second experiment, we aim to analyze whether a GNN trained on a small network generalizes well to a large network. We consider the same parameterization of the two-layer GNN in Table 1 and use the same training parameters of the first experiment. The GNN is trained to predict the ratings of the movie "Star Wars" on similarity networks with $n = 118, 203, 338, 603$, and 1682 nodes, where one of the nodes is always "Star Wars" and others are picked at

Table 2 Average RMSE Over Ten Random Data Splits for the Six Movies With Most Ratings in the Data Set

Parametrization	RMSE
Linear parametrization	1.967
Graph filter	1.054
FCNN	1.116
Graph perceptron, $L = 1$	1.079
GNN, $L = 2$	1.076
GNN, $L = 1, F = 64$	1.050
GNN, $L = 2, F_1 = 64, F_2 = 32$	**0.964**

Table 3 Average RMSE Achieved on the Graph Where the GNN Is Trained (n Nodes) and on the Full Movie Graph for the Movie "Star Wars." Average Relative RMSE Difference

Graph / n	118	203	338	603	1682
n nodes	0.829	0.818	0.863	0.866	0.873
Full graph	4.069	3.908	2.150	1.201	0.873
Difference	79.5%	79.0%	56.0%	23.4%	0.0%

random. After training, each GNN is then tested on the full movie network.

Table 3 shows the average RMSEs obtained on both the graphs where the GNN was trained and the full movie graph for ten random data splits. It also shows the average relative difference between the RMSE on these graphs. We observe that the prediction error on the full movie network approaches the error realized on the trained network as n increases. These results suggest that GNNs are *transferable*, a property that we discuss, in more detail, in Section V.

IV. STABILITY PROPERTIES OF GNNs

Permutation equivariance is a fundamental property of graph filters (see Proposition 1) and GNNs (see Proposition 2) since it allows them to exploit the graph structure and, thus, generalize better to unseen samples coming from the same graph [39], [41]. However, graphs rarely exhibit perfect symmetries, as illustrated in Fig. 2, but rather show near permutation symmetries, as shown in Fig. 3.

Stability to graph support perturbations quantifies how much the output of the graph filter changes in relation to the size of the perturbation. That is, if the graph support has changed slightly (with respect to a permutation of itself), then the output of a trained graph filter or GNN will also change slightly [39]. This property is particularly important in graph data where the structure of the graph, described by $\mathbf{S}$, is generally given in the problem and might not be known precisely [60]. For example, in the problem of movie recommendation (see Section II-B), the edges of the graph are built based on the rating similarity between the items [see (5)]. Estimating this value depends on the training set, and thus, there is an error incurred in obtaining it. Therefore, we usually train over an inferred graph that is not exactly the true graph over which the data are actually defined. The stability property guarantees that the trained parameterization (either a graph filter or a GNN) will yield the expected performance as long as the estimation of the support is good enough [39].

In this section, we present the stability property of graph filters and GNNs for a relative perturbation model (see Section IV-A). Stability is, thus, another fundamental property that complements permutation equivariance, establishing the mechanisms by which graph filters and GNNs adequately exploit the graph structure to offer better generalization capabilities.

Both permutation equivariance and stability are properties shared by graph filters and GNNs, and thus, they explain their superior performance with respect to arbitrary linear transforms or FCNNs, as observed in the recommendation problem (see Section III-D). In this example, we further observe that GNNs perform better than graph filters. Herein, we leverage the stability theorems and the effect of nonlinearities to explain why GNNs perform better than graph filters. We show that nonlinearities have a demodulating effect on the frequency domain that allows GNNs to be simultaneously stable and discriminative, a feat that cannot be achieved by graph filters (see Section IV-B).

In what follows, we focus on undirected graphs and parameterizations given either by graph convolutional filters with F input features and G output features [see (21)] or by GNNs [see (23)]. We consider GNNs that satisfy the following assumptions.

Assumption 1 (GNN Architecture): Let Φ be a GNN parameterization (23) with the following architecture.

1) It consists of $L > 0$ layers.
2) It obtains F_l features at the output of each layer.
3) The graph filters [see (21)] are described by the tensor of coefficients $\mathbf{H} = \{\mathbf{H}_{lk}\}_{l,k}$, with $\mathbf{H}_{lk} \in \mathbb{R}^{F_{l-1} \times F_l}$.
4) The output of the filtering stage of each layer l satisfies $\|\mathbf{U}_l\| \leq B\|\mathbf{X}_{l-1}\|$ [see (21)] for some $B > 0$.
5) The chosen nonlinearity σ is normalized Lipschitz continuous, $|\sigma(a) - \sigma(b)| \leq |a - b|$ for $a, b \in \mathbb{R}$, and satisfies $\sigma(0) = 0$.

We note that Assumption 1 is made on the resulting trained GNN. Assumptions 1)–3) are determined by the hyperparameters of the architecture and, as such, are a design choice. Assumption 4) needs to be satisfied only on some finite interval $[\lambda_{\min}, \lambda_{\max}]$ and is always the case, in theory, for graph convolutional filters (21) with finite coefficients. In practical terms, some choices of $\mathbf{S}$ may lead to numerical instabilities when computing $\mathbf{S}^k$. There are several ways to address this, as discussed in [15]. Assumption 5) is satisfied by most of the commonly chosen nonlinearities ($\tanh$, ReLU, and sigmoid).

A. Relative Perturbations

Permutations are a very particular case of a modification or *perturbation* to which the graph support $\mathbf{S}$ can be subjected (see Fig. 2). We are interested, however, in more general perturbations $\hat{\mathbf{S}}$ (see Fig. 3) and in analyzing how the parameterization Φ changes under these perturbations of the graph support. To measure the change in the parameterization, and in light of the permutation equivariance property of Propositions 1 and 2, we define the operator distance modulo permutations.

Definition 1 (Operator Distance Modulo Permutations): Let $\mathbf{S}$ be the support matrix of a graph $\mathbf{G}$, and let $\hat{\mathbf{S}}$ be the support matrix of a perturbed graph $\hat{\mathbf{G}}$. Let $\mathbf{H}$ be the tensor of filter coefficients that describe the parameterization Φ [see (21) or (23)]. Then, the *operator distance modulo* *permutation* is defined as

$$\|\Phi(\cdot;\mathbf{H},\mathbf{S})-\Phi(\cdot;\mathbf{H},\hat{\mathbf{S}})\|_{\mathcal{P}} \\ = \min_{\mathbf{P}\in\mathcal{P}} \max_{\mathbf{X}:\|\mathbf{X}\|=1} \|\Phi(\mathbf{X};\mathbf{H},\mathbf{S})-\Phi(\mathbf{X};\mathbf{H},\mathbf{P}^T\hat{\mathbf{S}}\mathbf{P})\| \quad (28)$$

where, for any $\mathbf{U}\in\mathbb{R}^{n\times G}$, we define $\|\mathbf{U}\|=\sum_{g=1}^{G}\|\mathbf{u}^g\|_2$.

We note that $\mathcal{P}$ denotes the set of all possible permutations

$$\mathcal{P}=\{\mathbf{P}\in\{0,1\}^{n\times n}:\mathbf{P}\mathbf{1}=\mathbf{1},\mathbf{P}^T\mathbf{1}=\mathbf{1}\}. \quad (29)$$

The operator distance modulo permutations measures how much the output of the parameterization Φ changes for a unit-norm signal $\mathbf{X}$ that makes the difference maximum and for a permutation that makes the difference minimum. Note that, in terms of the operator distance in Definition 1, the permutation equivariance property (see Propositions 1 and 2) implies that

$$\|\Phi(\cdot;\mathbf{H},\mathbf{S})-\Phi(\cdot;\mathbf{H},\mathbf{P}^T\mathbf{S}\mathbf{P})\|_{\mathcal{P}}=0 \quad (30)$$

for both graph filters and GNN parameterizations of Φ.

To better analyze how the output of the parameterization Φ changes when the underlying graph is perturbed, we proceed in the graph frequency domain, as is customary in signal processing. To do this, we consider the eigendecomposition of the support matrix $\mathbf{S}=\mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ to be given by an orthonormal set of eigenvectors collected in the columns of $\mathbf{V}$. We define the graph Fourier transform (GFT) of a graph signal $\mathbf{X}$ as a projection of the signal onto the eigenvectors of the support matrix $\mathbf{S}$ [11], [21], [61], [62]

$$\tilde{\mathbf{X}}=\mathbf{V}^T\mathbf{X}. \quad (31)$$

Note that, since $\mathbf{V}$ is an orthonormal matrix, then the inverse GFT is immediately defined as $\mathbf{X}=\mathbf{V}\tilde{\mathbf{X}}$.

With this definition in place, we can compute the GFT of the graph filter output $\mathbf{U}=\sum_{k=0}^{\infty}\mathbf{S}^k\mathbf{X}\mathbf{H}_k$ [see (21)] as [12]

$$\tilde{\mathbf{U}}=\mathbf{V}^T\mathbf{U}=\sum_{k=0}^{\infty}\mathbf{\Lambda}^k\tilde{\mathbf{X}}\mathbf{H}_k \quad (32)$$

where, due to the diagonal nature of $\mathbf{\Lambda}$, we can obtain the GFT as a pointwise multiplication in the graph frequency domain, akin to the convolution theorem [63, Section 2.9.6], [22], [61]. To see this more clearly, consider the ith frequency component of $\mathbf{U}$ for the gth feature, that is, the element (i,g) of $\tilde{\mathbf{U}}$ that we denote as $[\tilde{\mathbf{U}}]_{ig}=\tilde{u}_i^g$. Then, we note that

$$\tilde{u}_i^g=\sum_{f=1}^{F}h^{fg}(\lambda_i)\tilde{x}_i^f \quad (33)$$

for $\tilde{x}_i^f$ the ith frequency component of the fth feature of the input, where $h^{fg}(\lambda_i)$ is the frequency response of the (f,g) graph convolutional filter in (21), evaluated at λ_i. We formally define the frequency response of a graph filter [see (21)].

Definition 2 (Graph Filter Frequency Response): Given a graph filter [see (21)] with a tensor of filter coefficients $\mathbf{H}=\{\mathbf{H}_k\}_k$, $\mathbf{H}_k\in\mathbb{R}^{F\times G}$, the frequency response of the graph filter is the set of $F\times G$ polynomial functions $h^{fg}(\lambda)$, with

$$h^{fg}(\lambda)=\sum_{k=0}^{K}h_k^{fg}\lambda^k \quad (34)$$

for a continuous variable λ, and where $h_k^{fg}=[\mathbf{H}_k]_{fg}$ is the (f,g)th element of $\mathbf{H}_k$, corresponding to the kth filter coefficient of the (f,g) graph convolutional filter in the corresponding filterbank.

Per Definition 2, the frequency response of a filter is a collection of polynomial functions characterized solely by the filter coefficients and so it is independent of the graph. The effect of the specific support matrix $\mathbf{S}$ on a graph filter is observed by instantiating the frequency response on the specific eigenvalues [see (33)]. However, the shape of the frequency response is actually independent of the graph and determined by the filter coefficients.

It is evident from (33) that the GFT of the output of a graph filter is a pointwise multiplication of the GFT of the input and the frequency response of the filter. An important distinction with traditional signal processing, however, is that the GFT of a signal depends on the eigenvectors of the support matrix $\mathbf{S}$, and the GFT of a filter depends on the eigenvalues of $\mathbf{S}$ [61], while, in traditional SP, the FT of both the signal and the filter only depends on the eigenvalues $e^{-j2\pi n/N}$.

We are particularly interested in filters that satisfy the integral Lipschitz condition. While traditional Lipschitz filters are those whose frequency response is Lipschitz continuous [39, Definition 2], *integral* Lipschitz filters are those that are Lipschitz continuous, but with a constant that depends on the midpoint of the values considered. See Fig. 7 for an illustrative comparison between Lipschitz filters and integral Lipschitz filters. We formally define integral Lipschitz filters as follows.

Definition 3 (Integral Lipschitz Graph Filters): Given a filter [see (21)] with a tensor of filter coefficients $\mathbf{H}=\{\mathbf{H}_k\}_k$ with $\mathbf{H}_k\in\mathbb{R}^{F\times G}$, we say that it is an *integral Lipschitz graph filter* if its frequency response [see Definition 2] satisfies

$$|h^{fg}(\lambda_1)-h^{fg}(\lambda_2)|\leq\frac{C}{|\lambda_1+\lambda_2|/2}|\lambda_1-\lambda_2| \quad (35)$$

for some $C>0$, and for all $\lambda_1,\lambda_2\in\mathbb{R},\lambda_1\neq\lambda_2$ and all $f=1,\dots,F$ and $g=1,\dots,G$.

Integral Lipschitz filters (see Definition 3) are those filters whose frequency response (see Definition 2) is Lipschitz continuous on continuous variable λ with a Lipschitz

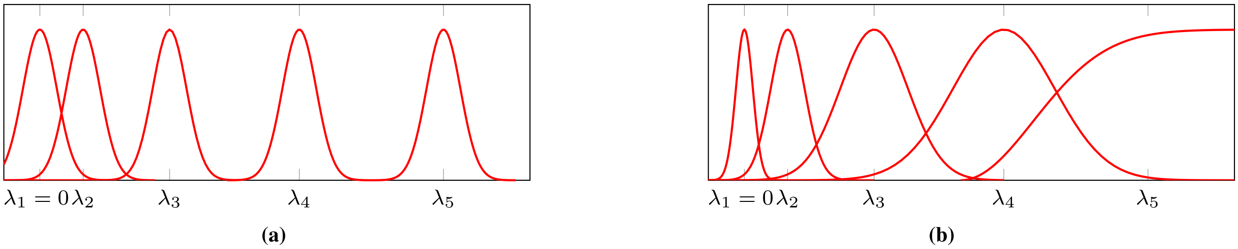


Fig. 7. ***Frequency response (see Definition 2) of bank of graph filters [see (21)]. (a) Lipschitz filter with $F = 1$ input feature and $G = 5$ output features. The frequency response of a Lipschitz filter has five functions of the form (34) and all satisfy Lipschitz continuity $|h^{fg}(\lambda_1) - h^{fg}(\lambda_2)| \leq C|\lambda_1 - \lambda_2|$. In this illustrative plot, this condition is met exactly. The minimum width of the functions (34) is determined by C since this value limits the maximum value of the derivative. The minimum width is the same throughout the spectrum. (b) Integral Lipschitz filter (see Definition 3) with $F = 1$ input feature and $G = 5$ output features. The frequency response of an integral Lipschitz filter has five functions of the form (34) and all satisfy (35). In this plot, this condition is met exactly. The minimum width of the functions (34) depends on their location in the spectrum since the maximum value of the derivative is bounded by $2C/|\lambda_1 + \lambda_2|$. Therefore, filters located in smaller eigenvalues (i.e., λ_1) can be narrower than filters located in larger eigenvalues (i.e., λ_5).***

constant that is inversely proportional to the midpoint of the interval. For example, if λ_1 or λ_2 is large, the resulting Lipschitz constant $2C/(\lambda_1 + \lambda_2)$ is small. This implies that these filters need to be flat for large values of λ (i.e., they do not change) but can be arbitrarily thin for values of λ near zero (i.e., they can change arbitrarily). See Fig. 7(b) for an example of an illustration of the frequency response of a graph filter that satisfies the integral Lipschitz condition. Note that (35) implies $|\lambda(h^{fg}(\lambda))'| \leq C$ for $(h^{fg}(\lambda))'$ being the derivative of $h^{fg}(\lambda)$. This condition is reminiscent of the scale invariance of wavelet filter banks [64], and there are several graph wavelet banks that satisfy it (see [65] and [66]).

To measure the distance between a graph $\mathbf{S}$ and its corresponding perturbation $\hat{\mathbf{S}}$, we adopt a *relative perturbation* model, which ties the changes of the graph to the underlying structure.

Definition 4 (Relative Perturbations): Given a support matrix $\mathbf{S}$ and a perturbed support $\hat{\mathbf{S}}$, define the *relative error set* as

$$\mathcal{E}(\mathbf{S}, \hat{\mathbf{S}}) = \Bigg\{ \mathbf{E} \in \mathbb{R}^{n \times n} : \mathbf{P}^T \hat{\mathbf{S}} \mathbf{P} = \mathbf{S} + \frac{1}{2}(\mathbf{SE} + \mathbf{ES}), \quad \mathbf{P} \in \mathcal{P}, \mathbf{E} = \mathbf{E}^T \Bigg\}. \tag{36}$$

The size of the *relative perturbation* is

$$d(\mathbf{S}, \hat{\mathbf{S}}) = \min_{\mathbf{E} \in \mathcal{E}(\mathbf{S}, \hat{\mathbf{S}})} \|\mathbf{E}\|. \tag{37}$$

The relative error set (36) is defined as the set of all symmetric error matrices $\mathbf{E}$ such that, when multiplied by the shift operator and added back to it, it yields a permutation of the perturbed support $\hat{\mathbf{S}}$. The relative perturbation size (37) is given by the minimum norm of all such relative error matrices and, thus, measures how close $\mathbf{S}$ and $\hat{\mathbf{S}}$ are to being permutations of each other, as determined by the multiplicative factor $\mathbf{E}$.

The relative perturbation model takes into consideration the structure of the graph when measuring the change in the support by tying the changes in the edge weights of the graph to its local structure. To see this, note that the difference between the edge weight $[\mathbf{S}]_{ij}$ of the original graph $\mathbf{S}$ and the corresponding edge $[\mathbf{P}_0^T \hat{\mathbf{S}} \mathbf{P}_0]_{ij}$ of the perturbed graph $\hat{\mathbf{S}}$ is given by the corresponding entry $[\mathbf{ES} + \mathbf{SE}]_{ij}$ of the perturbation factor $\mathbf{ES} + \mathbf{SE}$. It is ready to see that this quantity is proportional to the sum of the degrees of nodes i and j scaled by the entries of $\mathbf{E}$. As the norm of $\mathbf{E}$ grows, the entries of the graphs $\mathbf{S}$ and $\mathbf{P}_0^T \hat{\mathbf{S}} \mathbf{P}_0$ become more dissimilar. However, parts of the graph that are characterized by weaker connectivity change by amounts that are proportionally smaller to the changes that are observed in parts of the graph characterized by stronger links. This is in contrast to absolute perturbations where edge weights change by the same amount irrespective of the local topology of the graph.

Relative perturbations arise in many practical problems, and as a matter of fact, the diffeomorphism used in the seminal work by Mallat [18] can be modeled as a relative perturbation since each point in the Euclidean space is perturbed depending on the position of the point (i.e., it takes into account the original structure of the space). Most notable, though, is the case of covariance-based graphs, where the edge weights are a function of the correlation between the nodes. We typically estimate this correlation from a given data set, and this estimation incurs an error that is proportional to the true value of the correlation [67], [68]. Thus, the relationship between the estimate $\hat{\mathbf{S}}$ and the true graph $\mathbf{S}$ follows the relative perturbation model. We note that this is precisely the case in the problem of movie recommendation (see Section II-B), where perturbations arising from the imperfect estimation of the rating similarities (5) fall under the relative perturbation model.

Integral Lipschitz filters (see Definition 3) are stable to relative perturbations (see Definition 4) per the following theorem [39, Theorem 2].

Theorem 1 (Graph Filter Stability to Relative Perturbations): Let $\mathbf{S}$ and $\hat{\mathbf{S}}$ be the support matrices of a graph $\mathbf{G}$ and its perturbation $\hat{\mathbf{G}}$, respectively. Let Φ be a graph filter [see (21)] with a tensor of filter coefficients $\mathbf{H} = \{\mathbf{H}_k\}_k$, $\mathbf{H}_k \in \mathbb{R}^{F\times G}$. If Φ is an integral Lipschitz filter (see Definition 3) with $C > 0$ and if the relative perturbation size satisfies $d(\mathbf{S}, \hat{\mathbf{S}}) \leq \varepsilon$ (see Definition 4), then

$$\|\mathbf{\Phi}(\cdot;\mathbf{H},\mathbf{S}) - \mathbf{\Phi}(\cdot;\mathbf{H},\hat{\mathbf{S}})\|_{\mathcal{P}} \leq \varepsilon(1+\delta\sqrt{n})CG + \mathcal{O}(\varepsilon^2) \quad (38)$$

where $\delta = (\|\mathbf{U}-\mathbf{V}\|_2+1)^2-1$ is the *eigenvector misalignment constant* for $\mathbf{U}$, the eigenvector basis of the absolute error matrix $\mathbf{E}$ that solves (37).

Theorem 1 asserts that a change in the output of a graph filter caused by a relative perturbation of the graph support is upper bounded in proportion to the size of the perturbation (37). This property of stability to relative perturbations is inherited by GNNs, as shown in the following [39, Theorem 4].

Theorem 2 (GNN Stability to Relative Perturbations): Let $\mathbf{S}$ and $\hat{\mathbf{S}}$ be the support matrices of a graph $\mathbf{G}$ and its perturbation $\hat{\mathbf{G}}$, respectively. Let Φ be a GNN [see (23)] that satisfies Assumption 1. If the filters used in Φ are integral Lipschitz (see Definition 3) with $C > 0$ and if the relative perturbation size satisfies $d(\mathbf{S}, \hat{\mathbf{S}}) \leq \varepsilon$ (see Definition 4), then

$$\|\mathbf{\Phi}(\cdot;\mathbf{H},\mathbf{S}) - \mathbf{\Phi}(\cdot;\mathbf{H},\hat{\mathbf{S}})\|_{\mathcal{P}} \leq \varepsilon(1+\delta\sqrt{n})CB^{L-1}\prod_{l=1}^{L} F_l + \mathcal{O}(\varepsilon^2) \quad (39)$$

where $\delta = (\|\mathbf{U}-\mathbf{V}\|_2+1)^2-1$ is the *eigenvector misalignment constant* for $\mathbf{U}$, the eigenvector basis of the relative error matrix $\mathbf{E}$ that solves (37).

Theorem 2 states that the change in the output of the GNN caused by a relative perturbation of the graph support is upper bounded in a proportional manner to the size of the perturbation (37). Theorem 2, thus, complements Theorem 1, quantifying how the stability of graph filters gets inherited by GNNs.

The main conclusion and key takeaway of Theorems 1 and 2 are that the stability bound of both graph filters and GNNs is linear on the size of the perturbation, making both parameterizations stable to relative perturbations of the graph support. This bound also holds for all graphs with the same size n. We emphasize that this bound establishes Lipschitz continuity of graph filters and GNNs *with respect to changes in the underlying support*, not with respect to the input.[1] We further emphasize that the results in Theorems 1 and 2 hold for parameterizations using the same tensor filter coefficients $\mathbf{H}$. More specifically, stability to relative perturbations requires that the graph filters obtained after training be integral Lipschitz (see Definition 3). This condition is trivial on bounded support $[\lambda_{\min}, \lambda_{\max}]$ for filters given by an analytic frequency response (21). As a matter of fact, the actual value of C can be impacted during training by adding the integral Lipschitz condition (35) as a penalty on the loss function of the corresponding ERM problem (3).

The stability bound of Theorems 1 and 2 is proportional to the size of the perturbation. The proportionality constant is given by two terms. The first term is $(1+\delta\sqrt{n})$ and involves the eigenvector misalignment constant δ, which measures the change in the graph frequency basis caused by the perturbation. This term is given by the admissible perturbations of the specific problem under consideration. We note that, while δ provided here applies for any graph and any relative perturbation (see Definition 4), it is a coarse bound, which can be improved if we know that the space of possible perturbations is restricted by extraneous information, as is the case of Euclidean data [18]. For a numerical experiment showing how conservative the bound is, see [39, Fig. 6].

The second term is CG for graph filters or $CB^{L-1}\prod_{l=1}^{L} F_l$ for GNNs and is a direct consequence of the design choices that result in the specific graph filters used in the parameterization. The values of G or $\prod_{l=1}^{L} F_l$ are design choices, whereas the values of C and B result from the training phase. As discussed earlier, both these values can be impacted by an appropriate choice of penalty function during training if stability is to be increased. We note that the resulting filters can, thus, compensate for the specific perturbation characteristics.

Remark 3 (Absolute Perturbations): An alternative to the relative perturbation model is the absolute one [39]. In this case, the distance between $\mathbf{S}$ and $\hat{\mathbf{S}}$ is given by the norm of a matrix $\mathbf{E}$ such that we can write $\mathbf{P}\hat{\mathbf{S}}\mathbf{P}^T = \mathbf{S} + \mathbf{E}$ for some perturbation matrix $\mathbf{P}$. Note, however, that this model can be misleading, in which the graph structure can be altered completely without this being reflected in the value of ε. To see this, consider a stochastic block model with two disconnected communities. An absolute perturbation given by the identity matrix results in a perturbed graph that still respects this two-block structure. However, an absolute perturbation given by the antidiagonal identity matrix would disrupt this two-block structure by forcing connections between the blocks. Yet, both perturbations have the same absolute size ε. This is also evident in that the sparsity of the graph is completely lost. As we can see, absolute perturbations do not capture the specifics of the graph support they affect, so we choose to focus on relative perturbations. Details on the stability under absolute perturbation model can be found in [39].

Remark 4 (Computation of the Bound): The key contribution from Theorems 1 and 2 is that the change in the output of a GNN due to a change in the graph support is proportional to the size of the perturbation. This has

[1] GNNs and graph filters are also Lipschitz continuous with respect to the input, and this is trivial to show by using operator norms.

important implications in that a GNN trained on one graph can be used on another graph as long as the graphs are similar. This may entail computing $d(\mathbf{S}, \hat{\mathbf{S}})$ directly, which would lead to a combinatorial problem. To avoid this, we can estimate $d(\mathbf{S}, \hat{\mathbf{S}})$ by computing $\|\mathbf{S} - \hat{\mathbf{S}}\|/\|\mathbf{S}\|$. As for the proportionality constant, we emphasize that the stability of the architecture can be affected by changing the integral Lipschitz constant of the filter, which can be done through training. With respect to the eigenvector misalignment constant, knowing its exact value does not alter the conceptual implications of Theorems 1 and 2. This constant depends on the specific perturbation, and if more knowledge is available, it can be computed directly, as is the case of the diffeomorphism in [18]. Alternatively, more restrictions can be imposed on it [39, Theorem 3]. In any case, we note that $\delta \leq 8$ always holds since it is related to the norm of unitary matrices.

B. Discussion and Insights

Graph signals $\mathbf{X}$ can be completely characterized by their frequency content $\tilde{\mathbf{X}}$ given the one-to-one correspondence between the GFT and the inverse GFT [see (31)]. Therefore, to analyze, understand, and learn from signals, we need to use functions Φ that adequately capture the difference and similarities of signals throughout the frequency spectrum [61]. This concept is known in signal processing as filter discriminability and is concerned with how well a function Φ can tell apart different sections of the frequency spectrum.

In graphs, the spectrum is discrete and given by the eigenvalues $\lambda_1 < \cdots < \lambda_n$ of the graph support $\mathbf{S}$. Perturbations to the graph structure $\mathbf{S}$ alter the eigenvalues and, therefore, alter the location of the different frequency coefficients of the signal within the given spectrum. It is evident, then, that the concept of discriminability is related to the concept of stability since relevant parts of the spectrum that need to be told apart (discriminability) change under perturbations of the graph support (stability). Thus, to analyze both the discriminability and stability of a graph filter, we need to analyze the shape of its frequency response (see Definition 2).

Stability to relative perturbations (see Definition 4) requires integral Lipschitz filters (see Definition 3) as per Theorems 1 and 2. The maximum discriminability of integral Lipschitz filters, however, is not only determined by the integral Lipschitz constant C but also by the position in the spectrum. Recall that integral Lipschitz filters are Lipschitz with a constant $2C/(\lambda_1 + \lambda_2)$ that depends on the spectrum. Thus, if we are in a portion of the spectrum where λ is large, then the discriminability is very poor since the maximum derivative has to be almost zero, irrespective of C. On the contrary, if we are on the low-eigenvalue part of the spectrum, the discriminability can be arbitrarily high since the derivative of the frequency response can be arbitrarily large. In a way, the value of C helps determine the eigenvalue at which the integral Lipschitz filters enter the *flat* zone (larger C implies that larger eigenvalues can be discriminated before the filter becomes flat) but do not affect the overall discriminability for small eigenvalues. The value of C, however, does affect the stability of both graph filters and GNNs, where lower values of C means more stable representations (see Theorems 1 and 2).

This implies that, under the relative perturbation model, the discriminability of the filters is independent of their stability, meaning that, around low eigenvalues, they can be arbitrarily discriminative, while, at high eigenvalues, they cannot discriminate any frequency coefficient. All of these are irrespective of the value of C. This suggests that integral Lipschitz graph filters are well equipped to successfully learn from signals, as long as the relevant information is located in low-eigenvalue content. This limits their use of this specific class of signals. GNNs, however, can successfully capture information from high eigenvalues by leveraging the nonlinearity and the subsequent graph filters. This can be better understood by looking at a specific, illustrative, and conceptual example as we do next.

Consider the particular case of a perturbation that is given by an edge dilation, that is, $\hat{\mathbf{S}} = (1+\varepsilon)\mathbf{S}$, where $\varepsilon \approx 0$ is small. This is a particular instance of a relative perturbation model [see Definition 4]. In the case of the movie recommendation problem, this can happen if we use a biased estimator to compute the rating similarities, and thus, $\hat{\mathbf{S}}$, the graph on which we operate, is an edge dilation of the actual graph $\mathbf{S}$. Note that $\hat{\mathbf{S}}$ and $\mathbf{S}$ share the same eigenvectors so that the eigenvector misalignment constant of Theorems 1 and 2 is $\delta = 0$. The eigenvalues get perturbed as $\hat{\lambda}_i = (1+\varepsilon)\lambda_i$. This implies that larger eigenvalues get perturbed more than smaller eigenvalues.

In the context of this very simple edge dilation perturbation, we see in Fig. 8(a) an illustration that Lipschitz filters are not stable. This is because, for large eigenvalues, the change in the output of a filter is very large, even if the perturbation ε is small. To see this, notice that $|h(\hat{\lambda}_i) - h(\lambda_i)| \leq C|\hat{\lambda}_i - \lambda_i| = C\varepsilon\lambda_i$ so that, if λ_i is large, the difference in the filter output $|h(\hat{\lambda}_i) - h(\lambda_i)|$ can be very large, even if ε is small.

In contrast, integral Lipschitz filters are stable, as illustrated in Fig. 8(b). For small eigenvalues, these filters can have arbitrary variations, but, since small ε does not cause a big change in the eigenvalues, and the output is similar. For large eigenvalues, the frequency response is flat; thus, even if there is a high variability of the eigenvalues, the filter output remains constant. This follows from the integral Lipschitz condition, where $|h(\hat{\lambda}_i) - h(\lambda_i)| \leq 2C|\hat{\lambda}_i - \lambda_i|/|\hat{\lambda}_i + \lambda_i| \approx 2C\varepsilon$ only depends on ε but not on the specific eigenvalue, leading to stability.

The price that integral Lipschitz filters pay for stability is that they cannot discriminate information located at high eigenvalues. Consider that we want to tell apart two single-feature signals, $\mathbf{x} = \mathbf{v}_n$ and $\mathbf{y} = \mathbf{v}_{n-1}$, where $\mathbf{v}_i$ is the eigenvector associated with λ_i (or $\hat{\lambda}_i$ in the perturbed graph). As we can see on the illustration in Fig. 9(a), this

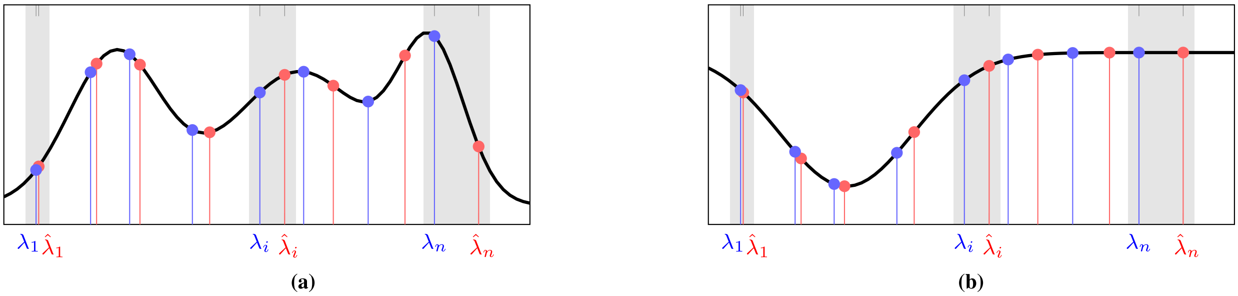


Fig. 8. ***Effect of a graph dilation $\hat{\mathbf{S}} = (1+\varepsilon)\mathbf{S}$. The eigenvalues move from λ_i (in blue) to $\hat{\lambda}_i = (1+\varepsilon)\lambda_i$ (in red). Even if $\varepsilon \approx 0$, large eigenvalues change more than small eigenvalues. (a) Lipschitz filters are not stable. A small perturbation causes a large change in the output of the filter due to the large change in large eigenvalues. (b) Integral Lipschitz filters are stable. For small eigenvalues, the filter can change, but the eigenvalues do not change much. For large eigenvalues, the filter is flat, and thus, the large change in eigenvalues still yields the same output.***

is not doable by means of integral Lipschitz filters. On the contrary, we could easily discriminate between these two signals by using Lipschitz filters, as illustrated in Fig. 9(b). However, this leads to an unstable filter, as discussed before. Therefore, when using linear graph filters as parameterizations Φ, we are faced with the tradeoff between discriminability and stability (where we need to increase the C of integral Lipschitz filters to achieve discriminability at high eigenvalues) or, alternatively, stick to processing graph signals whose relevant information is located on low eigenvalues.

GNNs are stable under relative perturbations by employing integral Lipschitz filters (see Theorem 2). While, as discussed above, integral Lipschitz filters are unable to discriminate information located in high eigenvalues, GNNs can do so by leveraging the pointwise nonlinearity. Essentially, applying a nonlinearity to a signal spreads its information content throughout the spectrum, creating frequency content in locations where it was not before. As we can see in the illustration in Fig. 10(a), the frequency content of $\mathbf{x} = \mathbf{v}_n$ after applying the nonlinearity is located throughout the frequency spectrum. The same happens when applying σ to $\mathbf{y} = \mathbf{v}_{n-1}$, as shown in the illustration in Fig. 10(b). Even more so, the resulting frequency content is different in both resulting signals. Once the frequency content has been spread throughout the spectrum, the integral Lipschitz graph filters can, indeed, discriminate between these two signals by processing only the low-eigenvalue frequency content. In essence, the nonlinearities in GNNs act as frequency demodulators, spreading the information content throughout the spectrum. This allows for subsequent filters to process this information in a stable manner. Thus, GNNs improve on graph filters, by processing information in a way that is simultaneously discriminative and stable.

V. TRANSFERABILITY OF GNNs

In different instances of the same network problem, it is not uncommon for different graphs, even of different sizes, to “look similar” in the sense that they share certain defining structural characteristics. This motivates studying groups of graphs—or *graph families*—and investigating whether graph filters and GNNs are transferable within them. Transferability of information processing

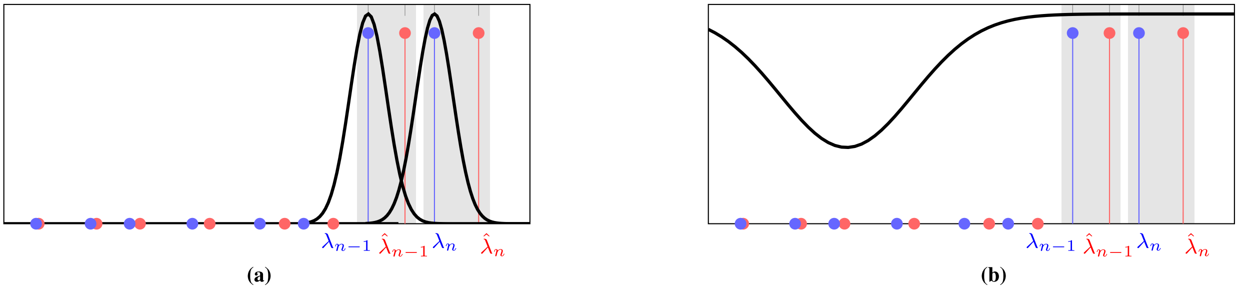
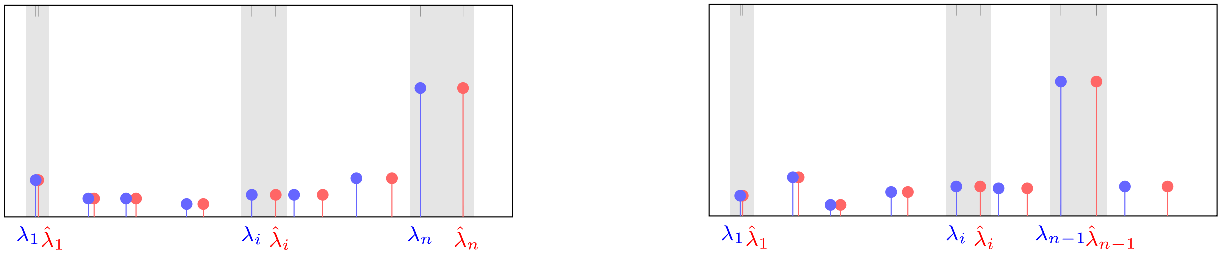


Fig. 9. ***Discriminability of large eigenvalues. Let $\mathbf{x} = \mathbf{v}_n$ and $\mathbf{y} = \mathbf{v}_{n-1}$ be two different signals that we want to discriminate. (a) This can be done by using a Lipschitz graph filter with $G = 2$ output features and a reasonable value of C. However, if the graph is subject to an edge dilation, then the eigenvalues will fall out of the passband of the frequency response and, thus, yield an output of zero. Therefore, Lipschitz filters can discriminate signals with large eigenvalue content but cannot do so in a stable manner. (b) Integral Lipschitz filter is not able to discriminate between $\mathbf{x}$ and $\mathbf{y}$ since it cannot be narrow for large eigenvalues (unless the integral Lipschitz constant C is very large, compromising the stability). In summary, Lipschitz filters can discriminate large eigenvalue content but are not stable, while integral Lipschitz filters are stable but cannot discriminate large eigenvalue content.***



(a) **(b)**

Fig. 10. ***Effect of applying nonlinearities. (a) Frequency content of signal $\sigma(x) = \text{ReLU}(v_n)$. (b) Frequency content of signal $\sigma(y) = \text{ReLU}(v_{n-1})$. The use of nonlinearities creates frequency content in parts of the spectrum that there were none. The nonlinearity spreads the frequency content throughout the spectrum, in an effect akin to demodulation. This is a fundamental contribution of nonlinearities since the frequency content at low eigenvalues can be stably discriminated by the graph filters used in the following layer. While we cannot control what shape the signal will have after being applied a nonlinearity, we observe that this content will likely be different, and thus, will be further discriminated. The effect of nonlinearities allows GNNs to process content in large eigenvalues in a stable manner (by spreading it into low eigenvalues). (a) $[\text{ReLU}(x)]_i = \max\{0, [x]_i\}$. (b) $[\text{ReLU}(y)]_i = \max\{0, [y]_i\}$.***

architectures is important because it allows the reuse of systems without the need to retrain or redesign. This is especially useful in applications where the network size is dynamic, for example, recommendation systems for a growing product portfolio (see Sections II-B and III-D).

From the architecture perspective, transferability is akin to replacing the graph with another graph in the same family, which, in itself, is a kind of perturbation. Therefore, transferability can be seen as a type of stability. In this section, we analyze the transferability of graph filters and GNNs in a similar fashion to Section IV, with particular focus on families of undirected graphs identified by objects called *graphons*. All analyses assume the multilayer and single-feature architectures of Section III-B.

A. Graphons and Graph Families

Graphons are bounded, symmetric, and measurable functions $\mathbf{W} : [0,1]^2 \rightarrow [0,1]$, which can be thought of as representations of undirected graphs with an uncountable number of nodes. An example is the exponential graphon $\mathbf{W}(u,v) = \exp(-\beta(u-v)^2)$ with parameter $\beta > 0$. Assigning nodes i and j to points u_i and u_j of the unit interval, the weight of the edge (i,j) is given by $\mathbf{W}(u_i,u_j)$. This weight is largest when u_i is close to u_j; therefore, the exponential graphon can be used to model graphs with cyclic or ring structure. As suggested by their infinite-dimensional structure, graphons are also the limit objects of convergent sequences of graphs.

A convergent sequence of graphs, denoted as $\{\mathbf{G}_n\}$, is characterized by the convergence of the density of certain structures, or *motifs*, in the graphs $\mathbf{G}_n$. We define these motifs as graphs $\mathbf{F} = (V', E')$ that are unweighted and undirected. Homomorphisms of $\mathbf{F}$ into $\mathbf{G} = (V, E, \mathbf{S})$ are defined as adjacency preserving maps. There are $|V|^{|V'|} = n^{n'}$ maps from V' to V, but only some of them are homomorphisms. Hence, we can define a density of homomorphisms $t(\mathbf{F}, \mathbf{G})$, which represents the relative frequency with which the motif $\mathbf{F}$ appears in $\mathbf{G}$.

Homomorphisms of graphs into graphons are defined analogously and denoted as $t(\mathbf{F}, \mathbf{W})$ for a motif $\mathbf{F}$ and a graphon $\mathbf{W}$. The graph sequence $\{\mathbf{G}_n\}$ converges to the graphon $\mathbf{W}$ if, for all finite, unweighted, and undirected graphs $\mathbf{F}$:

$$\lim_{n\to\infty} t(\mathbf{F}, \mathbf{G}_n) = t(\mathbf{F}, \mathbf{W}). \tag{40}$$

All graphons are limit objects of convergent graph sequences, and every convergent graph sequence converges to a graphon [45, Chapter 11]. This allows associating graphons with *families* of graphs of different sizes that share structural similarities. The simplest examples of such graphs are those obtained by evaluation of $\mathbf{W}$. In particular, our transferability results will hold for *deterministic graphs* $\mathbf{G}_n$ constructed by associating the regular partition $u_i = (i-1)/n$ to nodes $1 \leq i \leq n$ and the weights $\mathbf{W}(u_i, u_j)$ to edges (i,j). Explicitly

$$[\mathbf{S}_n]_{ij} = s_{ij} = \mathbf{W}(u_i, u_j) \tag{41}$$

where $\mathbf{S}_n$ is the adjacency matrix of $\mathbf{G}_n$. This sequence of deterministic graphs satisfies the condition in (40) and, therefore, converges to the graphon $\mathbf{W}$ [45, Chapter 11]. The convergence mode in (40) also allows for other, more general graph sequences than those consisting of deterministic graphs.

B. Graphon Filters

To understand the behavior of data that may be supported on the graphs belonging to a graphon family, it is also natural to consider the abstractions of *graphon data* and *graphon information processing systems*. Graphon data, or graphon signals, are defined as functions $X : [0,1] \rightarrow \mathbb{R}$ of L^2. These signals can be modified through graphon operations parameterized by the integral operator

$$(T_{\mathbf{W}}X)(v) := \int_0^1 \mathbf{W}(u,v)X(u)du \tag{42}$$

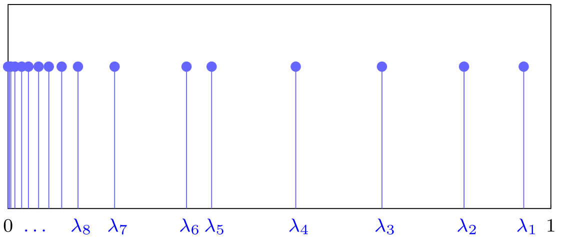


Fig. 11. ***Graphon eigenvalues. A graphon has an infinite number of eigenvalues λ_j but for any fixed constant c the number of eigenvalues $|\lambda_j| > c$ is finite. Thus, eigenvalues accumulate at 0, and this is the only accumulation point for graphon eigenvalues.***

which is called *graphon shift operator* (WSO) in analogy with the GSO [50]. Because $\mathbf{W}$ is bounded and symmetric, the WSO is a self-adjoint Hilbert–Schmidt operator, allowing us to express $\mathbf{W}$ in the operator's spectral basis—the *graphon spectra*—as

$$\mathbf{W}(u,v) = \sum_{i\in\mathbb{Z}\setminus\{0\}} \lambda_i \varphi_i(u)\varphi_i(v). \tag{43}$$

The operator $T_{\mathbf{W}}$ can, thus, be rewritten as

$$(T_{\mathbf{W}}X)(v) = \sum_{i\in\mathbb{Z}\setminus\{0\}} \lambda_i\varphi_i(v)\int_0^1 \varphi_i(u)X(u)du \tag{44}$$

where λ_i are the graphon eigenvalues, φ_i are the graphon eigenfunctions, and $i \in \mathbb{Z}\setminus\{0\}$. The eigenvalues are ordered according to their sign and in decreasing order of absolute value, that is, $1 \geq \lambda_1 \geq \lambda_2 \geq \cdots \geq \cdots \geq \lambda_{-2} \geq \lambda_{-1} \geq -1$. The eigenvalues accumulate around 0 as $|i| \to \infty$, as depicted in Fig. 11 [69, Theorem 3 and Chapter 28].

Graphon convolutions are defined as shift-and-sum operations where the shift is implemented by the graphon shift operator. Explicitly, a graphon convolutional filter is given by

$$\Phi(X;\mathbf{h},\mathbf{W}) = \sum_{k=0}^{K} h_k(T_{\mathbf{W}}^{(k)}X)(v) = (T_{\mathbf{H}}X)(v) \quad \text{with}$$
$$(T_{\mathbf{W}}^{(k)}X)(v) = \int_0^1 \mathbf{W}(u,v)(T_{\mathbf{W}}^{(k-1)}X)(u)du \tag{45}$$

where $T_{\mathbf{W}}^{(0)} = \mathbf{I}$ is the identity operator [50]. The vector $\mathbf{h} = [h_0,\ldots,h_K]$ collects the filter coefficients. Using the spectral decomposition in (44), $\Phi(X;\mathbf{h},\mathbf{W})$ can also be written as

$$\Phi(X;\mathbf{h},\mathbf{W}) = \sum_{i\in\mathbb{Z}\setminus\{0\}}\sum_{k=0}^{K} h_k\lambda_i^k\varphi_i(v)\int_0^1\varphi_i(u)X(u)du$$
$$= \sum_{i\in\mathbb{Z}\setminus\{0\}} h(\lambda_i)\varphi_i(v)\int_0^1\varphi_i(u)X(u)du. \tag{46}$$

Note that the spectral representation of $\Phi(X;\mathbf{h},\mathbf{W})$ is given by $h(\lambda) = \sum_{k=0}^{K} h_k\lambda^k$, which only depends on the graphon eigenvalues and on the coefficients h_k.

1) Generating Graph Filters From Graphon Filters: Like the spectral representation of the graphon filter, the spectral representation of the graph filter, as shown in Definition 2, depends uniquely on the graph eigenvalues and the filter coefficients. This allows making the coefficients h_k in (34) and (46) the same. Put differently, graphon filters can serve as generating models for graph filters on graphs evaluated from the graphon. Take the graphon filter $\Phi(X;\mathbf{h},\mathbf{W})$ from (45) and construct a partition $u_i = (i-1)/n$, $1 \leq i \leq n$, of $[0,1]$. The graph filter $\Phi(\mathbf{x}_n;\mathbf{h},\mathbf{S}_n) = \sum_{k=0}^{K} h_k\mathbf{S}_n^k\mathbf{x}_n$ can be obtained by defining

$$[\mathbf{S}_n]_{ij} = \mathbf{W}(u_i,u_j) \quad \text{and}$$
$$[\mathbf{x}_n]_i = X(u_i) \tag{47}$$

where $\mathbf{S}_n$ is the GSO of $\mathbf{G}_n$, the deterministic graph obtained from $\mathbf{W}$ as in equation (41), and $\mathbf{x}_n$ is the corresponding *deterministic graph signal* obtained by evaluating X at u_i.

Generating graph filters from graphon filters is helpful because it allows designing filters on graphons and applying them to graphs. This decouples the filter design from a specific graph realization. Conversely, it is also possible to define graphon filters induced by graph filters. The graphon filter induced by $\Phi(\mathbf{x}_n;\mathbf{h},\mathbf{S}_n) = \sum_{k=0}^{K} h_k\mathbf{S}_n^k\mathbf{x}_n$ is given by

$$\Phi(X_n;\mathbf{h},\mathbf{W}_n) = \sum_{k=0}^{K} h_k(T_{\mathbf{W}_n}^{(k)}X_n)(v) = \quad \text{with}$$
$$(T_{\mathbf{W}_n}^{(k)}X_n)(v) = \int_0^1 \mathbf{W}_n(u,v)(T_{\mathbf{W}_n}^{(k-1)}X_n)(u)du \tag{48}$$

where the graphon $\mathbf{W}_n$ is the *graphon induced by* $\mathbf{G}_n$ and X_n is the *graphon signal induced by the graph signal* $\mathbf{x}_n$, that is,

$$\mathbf{W}_n(u,v) = [\mathbf{S}_n]_{ij} \times \mathbb{I}(u\in I_i)\mathbb{I}(v\in I_j) \quad \text{and}$$
$$X_n(u) = [\mathbf{x}_n]_i \times \mathbb{I}(u\in I_i). \tag{49}$$

This definition allows comparing graph and graphon filters directly and analyzing the transferability of graph filters to graphs of different sizes.

2) Approximating Graph Filters With Graphon Filters: Consider graph filters obtained from a graphon filter, as in (47). For increasing n, $\mathbf{G}_n$ converges to $\mathbf{W}$, which means that these graph filters become increasingly similar to the graphon filter itself. Thus, the graph filter $\Phi(\mathbf{x}_n;\mathbf{h},\mathbf{S}_n)$ can be used to approximate $\Phi(X;\mathbf{h},\mathbf{W})$. In Theorem 3, we quantify how good this approximation is for different values of n. Because the continuous output $Y = \Phi(X; \mathbf{h}, \mathbf{W})$ cannot be compared with the discrete output $\mathbf{y}_n = \Phi(\mathbf{x}_n; \mathbf{h}, \mathbf{S}_n)$ directly, we consider the output of the graphon filter induced by $\Phi(\mathbf{x}_n; \mathbf{h}, \mathbf{S}_n)$, which is given by $Y_n = \Phi(X_n; \mathbf{h}, \mathbf{W}_n)$ [see (49)]. We also consider the following definitions and assumptions.

Definition 5 (c-Band Cardinality of $\mathbf{G}_n$): The c-band cardinality of $\mathbf{G}_n$, denoted as B_{nc}, is the number of eigenvalues λ_i^n of $\mathbf{W}_n$ with absolute value larger or equal to c, that is,

$$B_{nc} = \#\{\lambda_i^n \ : \ |\lambda_i^n| \geq c\}.$$

Definition 6 (c-Eigenvalue Margin of $\mathbf{G}_n$): The c-eigenvalue margin of $\mathbf{G}_n$, denoted as δ_{nc}, is given by

$$\delta_{nc} = \min_{i,j\neq i}\{|\lambda_i^n - \lambda_j| \ : \ |\lambda_i^n| \geq c\}$$

where λ_i^n and λ_i are the eigenvalues of $\mathbf{W}_n$ and $\mathbf{W}$, respectively.

Assumption 2: The graphon $\mathbf{W}$ is A_1-Lipschitz, that is, $|\mathbf{W}(u_2, v_2) - \mathbf{W}(u_1, v_1)| \leq A_1(|u_2 - u_1| + |v_2 - v_1|)$.

Assumption 3: The spectral response of the convolutional filter, h, is A_2-Lipschitz and nonamplifying, that is, $|h(\lambda)| < 1$.

Assumption 4: The graphon signal X is A_3-Lipschitz.

Theorem 3 (Graphon Filter Approximation by Graph Filter): Consider the graphon filter given by $Y = \Phi(X; \mathbf{h}, \mathbf{W})$, as in (46), where $h(\lambda)$ is constant for $|\lambda| < c$ [see Fig. 13]. For the graph filter instantiated from $\Phi(X; \mathbf{h}, \mathbf{W})$ as $\mathbf{y}_n = \Phi(\mathbf{x}_n; \mathbf{h}, \mathbf{S}_n)$ [see (47)], under Assumptions 2–4, it holds

$$\|Y - Y_n\|_{L_2} \leq \sqrt{A_1}\left(A_2 + \frac{\pi B_{nc}}{\delta_{nc}}\right) n^{-\frac{1}{2}} \|X\|_{L_2} + \frac{2A_3}{\sqrt{3}} n^{-\frac{1}{2}}$$

where $Y_n = \Phi(X_n; \mathbf{h}, \mathbf{W}_n)$ is the graph filter induced by $\mathbf{y}_n = \Phi(\mathbf{x}_n; \mathbf{h}, \mathbf{S}_n)$ [see (49)].

Theorem 3 gives an asymptotic upper bound to the error incurred when approximating graphon filters with graph filters. This bound depends on the filter transferability constant $\sqrt{A_1}(A_2 + \pi B_{nc}/\delta_{nc})n^{-0.5}$, which multiplies $\|X\|$, and on a fixed error term depending on the variability A_3 of X (see Assumption 4) and corresponding to the difference between X and the graphon signal X_n, which is induced by $\mathbf{x}_n$. For large n, the first term dominates the second. Hence, the quality of the approximation is closely related to the transferability constant.

Aside from decreasing asymptotically with n, the transferability constant depends on the graphon and on the filter parameters. The dependence on the graphon is due to A_1, which is proportional to the graphon variability (see Assumption 2). The dependence on the filter parameters happens through the constants A_2, B_{nc}, and δ_{nc}. The first two determine the variability of the filter's spectral response, which is controlled by both the Lipschitz constant A_2 (see Assumption 3) and the length of the band $[c, 1]$, as depicted in Fig. 13. In particular, the number of eigenvalues within this band, given by B_{nc}, should satisfy $B_{nc} \ll n$ (i.e., $B_{nc} < \sqrt{n}$). This restriction on the length of the passing band, which is necessary for asymptotic convergence, is a consequence of two facts. The first is that the eigenvalues of the graph converge to those of the graphon [45, Ch. 11.6], as illustrated in Fig. 12. The second is that the eigenvalues of the graphon, when ordered in decreasing order of absolute value, accumulate near zero. Combined, these facts imply that, for small eigenvalues, the graph eigenvalues are hard to match to the corresponding graphon eigenvalues, making consecutive eigenvalues difficult to discriminate. As a consequence, filters h with large variation near zero (i.e., small c) may modify matching graphon and graph eigenvalues differently, leading to large approximation error. Finally, note that when the $B_{nc} < \sqrt{n}$ requirement is satisfied, asymptotic convergence is guaranteed by convergence of the eigenvalues of $\mathbf{W}_n$ to those of $\mathbf{W}$ because $\delta_{nc} \rightarrow \min_{i\,:\,\lambda_i^n \geq c} |\lambda_i - \lambda_{i+\text{sgn(i)}}| \neq 0$, that is, δ_{nc} converges to the minimum eigengap of the graphon in the passing band.

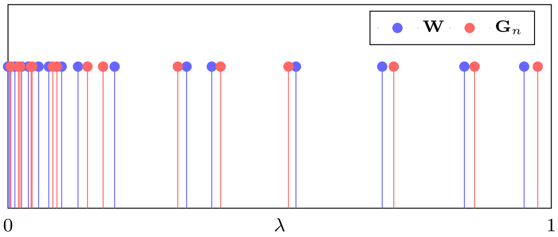


Fig. 12. ***Comparison of graphon eigenvalues (blue) and eigenvalues of a graph G$_n$ from a convergent graph sequence (red). As the number of nodes n grows, the eigenvalues of G$_n$ converge to those of W.***

C. Graph Filter Transferability

By the triangle inequality, transferability of graph filters follows directly from Theorem 3.

Theorem 4 (Graph Filter Transferability): Let $\mathbf{G}_{n_1}$ and $\mathbf{G}_{n_2}$, and $\mathbf{x}_{n_1}$ and $\mathbf{x}_{n_2}$, be graphs and graph signals obtained from the graphon $\mathbf{W}$ and the graphon signal X, as in (47), with $n_1 \neq n_2$. Consider the graph filters given by $\mathbf{y}_{n_1} = \Phi(\mathbf{x}_{n_1}; \mathbf{h}, \mathbf{S}_{n_1})$ and $\mathbf{y}_{n_2} = \Phi(\mathbf{x}_{n_2}; \mathbf{h}, \mathbf{S}_{n_2})$, and let their shared spectral response $h(\lambda)$ [see (34)] be constant for $|\lambda| < c$ [see Fig. 13]. Then, under Assumptions 2–4, it holds

$$\begin{aligned}\|Y_{n_1} - Y_{n_2}\|_{L_2} &\\ \leq \sqrt{A_1}&\left(A_2 + \frac{\pi B_c}{\delta_c}\right)\left({n_1}^{-\frac{1}{2}} + {n_2}^{-\frac{1}{2}}\right)\|X\|_{L_2}\\ &+ \frac{2A_3}{\sqrt{3}}\left(n_1^{-\frac{1}{2}} + n_2^{-\frac{1}{2}}\right)\end{aligned}$$

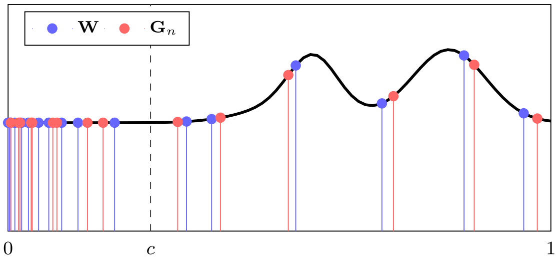


Fig. 13. *Lipschitz continuous filter with spectral response h(λ) constant for $\lambda < c$. The constant band for $\lambda \in [0, c]$ ensures that the filter has the same response for eigenvalues close to zero, which are harder to discriminate. This is necessary to avoid mismatch of the filter response for the graphon and graph eigenvalues in this range.*

where $Y_{n_j} = \Phi(X_{n_j}; \mathbf{h}, \mathbf{W}_{n_j})$ is the graphon filter induced by $\mathbf{y}_{n_j} = \Phi(\mathbf{x}_{n_j}; \mathbf{h}, \mathbf{S}_{n_j})$ [see (49)], $B_c = \max\{B_{n_1\, c}, B_{n_2\, c}\}$ [see Definition 5], and $\delta_c = \min\{\delta_{n_1\, c}, \delta_{n_2\, c}\}$ [see Definition 6].

Theorem 4 upper bounds the difference between the outputs of two identical graph filters on different graphs belonging to the same graphon family. Because this bound decreases asymptotically with n_1 and n_2, a filter designed for one of these graphs can be transferred to the other with good performance guarantees for large n_1 and n_2. Beyond values of n_1 and n_2 satisfying a specific error requirement of, say, ϵ, graph filters are scalable in the sense that they can be applied to any other graph with size $n > \max(n_1, n_2)$ and achieve less than ϵ error.

The transferability constant in Theorem 4 is equal to the sum of the transferability constant in Theorem 3 for $n = n_1$ and $n = n_2$. Even if Theorem 4 does not require explicitly defining the graphon filter and comparing its spectral response to that of the graph filters, the band $[c, 1]$ should be small to guarantee that the filter be able to match the eigenvalues of $\mathbf{G}_1$ and $\mathbf{G}_2$ and distinguish between consecutive eigenvalues [see Fig. 13]. Therefore, there exists a tradeoff between the transferability and discriminability of graph filters.

D. Graphon Neural Networks

The graphon neural network (WNN) is defined as the limit architecture of a GNN defined on the graphs of a convergent graph sequence. Denoting the nonlinear activation function σ, the ℓth layer of a multilayer WNN with $F_\ell = 1$ feature per layer (like the GNNs in Section III-B) is given by

$$X_\ell = \sigma\left(\Phi(X_{\ell-1}; \mathbf{h}_\ell, \mathbf{W})\right) \tag{50}$$

for $1 \leq \ell \leq L$. Note that the input signal at the first layer, X_0, is the input data X, and the WNN output is given by $Y = X_L$.

Similar to the GNN, this WNN can also be written as a map $Y = \Phi(X; \mathbf{H}, \mathbf{W})$, where the matrix $\mathbf{H} = \{\mathbf{h}_\ell\}_\ell$ groups the filter coefficients of all layers. Note that the parameters in $\mathbf{H}$ are completely independent of the graphon, which is another characteristic that WNNs have in common with GNNs.

1) Generating GNNs From WNNs: An important consequence of the GNN and WNN parameterizations is that, in the maps $\Phi(\mathbf{x}; \mathbf{H}, \mathbf{S})$ and $\Phi(X; \mathbf{H}, \mathbf{W})$, the parameters $\mathbf{H}$ can be the same. This allows sampling or evaluating GNNs from a WNN, that is, the WNN acts as a generating model for GNNs. To see this, consider the WNN $\Phi(X; \mathbf{H}, \mathbf{W})$ and define a partition $u_i = (i-1)/n$, $1 \leq i \leq n$, of $[0, 1]$. A GNN $\Phi(\mathbf{x}_n; \mathbf{H}, \mathbf{S}_n)$ can be obtained by evaluating the deterministic graph $\mathbf{G}_n$ and the deterministic graph signal $\mathbf{x}_n$, as in (47).

The interpretation of GNNs as instantiations of a WNN is important because it explicitly disconnects the GNN architecture from the graph. In this interpretation, the graph is not a fixed hyperparameter of the GNN but a parameter that can be changed according to the underlying graphon and the value of n. This reveals the ability of GNNs to *scale*. It also allows GNNs to be adapted both by optimizing the weights in $\mathbf{H}$ and by changing the graph $\mathbf{G}_n$, which adds degrees of freedom to the architecture at no additional computational cost.

WNNs induced by GNNs can also be defined. The WNN induced by a GNN $\Phi(\mathbf{x}_n; \mathbf{H}, \mathbf{S}_n)$ is given by $\Phi(X_n; \mathbf{H}, \mathbf{W}_n)$, where $\mathbf{W}_n$, the graphon induced by $\mathbf{G}_n$, and X_n, the graphon signal induced by $\mathbf{x}_n$, are as in (49). This definition allows establishing a direct comparison both between GNNs and WNNs and between GNNs on graphs of different sizes.

2) Approximating WNNs With GNNs: For large n, we can expect the GNNs instantiated from a WNN to become closer to the WNN itself at a similar rate at which the graphs $\mathbf{G}_n$ converge to $\mathbf{W}$. As such, the outputs of the GNN and WNN maps $\Phi(\mathbf{x}_n; \mathbf{H}, \mathbf{S}_n)$ and $\Phi(X; \mathbf{H}, \mathbf{W})$ should also grow closer, allowing the GNN to be used as a proxy for the WNN. To evaluate the quality of this approximation for different values of n, the outputs of $\Phi(\mathbf{x}_n; \mathbf{H}, \mathbf{S}_n)$ and $\Phi(X; \mathbf{H}, \mathbf{W})$ must be compared. This is done by considering the WNN induced by $\Phi(\mathbf{x}_n; \mathbf{H}, \mathbf{S}_n)$ and given by $Y_n = \Phi(X_n; \mathbf{H}, \mathbf{W}_n)$ [see (49)]. Under Assumption 5, the following theorem from [40] holds.

Assumption 5: The activation functions are normalized Lipschitz, that is, $|\sigma(x) - \sigma(y)| \leq |x - y|$, and $\sigma(0) = 0$.

This assumption is satisfied for most conventional nonlinearities, for example, ReLU and hyperbolic tangent.

Theorem 5 (WNN Approximation by GNN): Consider the L-layer WNN given by $Y = \Phi(X; \mathbf{H}, \mathbf{W})$, where $F_\ell = 1$ for $1 \leq \ell \leq L$. Let the graphon convolutions $h(\lambda)$ [see (46)] be such that $h(\lambda)$ is constant for $|\lambda| < c$ [see Fig. 13]. For the GNN instantiated from this WNN as $\mathbf{y}_n = \Phi(\mathbf{x}_n; \mathbf{H}, \mathbf{S}_n)$ [see (47)], under Assumptions 2–5, it holds

$$\|Y_n - Y\|_{L_2} \leq L\sqrt{A_1}\left(A_2 + \frac{\pi B_{nc}}{\delta_{nc}}\right) n^{-\frac{1}{2}} \|X\|_{L_2} + \frac{A_3}{\sqrt{3}} n^{-\frac{1}{2}}$$

where $Y_n = \Phi(X_n; \mathbf{H}, \mathbf{W}_n)$ is the WNN induced by $\mathbf{y}_n = \Phi(\mathbf{x}_n; \mathbf{H}, \mathbf{S}_n)$ [see (49)].

Given a graph $\mathbf{G}_n$ and a signal $\mathbf{x}_n$ obtained from $\mathbf{W}$ and X, as in (47), the GNN $\Phi(\mathbf{x}_n; \mathbf{H}, \mathbf{S}_n)$ can approximate the WNN $\Phi(\mathbf{X}; \mathbf{H}, \mathbf{W})$ with an error that decreases asymptotically with n. This error is upper bounded by a term proportional to the input, controlled by the *transferability constant* $L\sqrt{A_1}\,(A_2 + (\pi B_{nc})/\delta_{nc})\,n^{-0.5}$, and by a fixed error term given by $A_3/\sqrt{3n}$. The fixed error term is a truncation error due to "discretizing" X to obtain $\mathbf{x}_n$. Besides the dependence on the graphon and on the filter parameters, the transferability constant also depends on L. As for the constants A_1, A_2, B_{nc}, and δ_{nc}, the same comments as in Theorem 3 apply.

E. GNN Transferability

By Theorem 5 and the triangle inequality, the following theorem from [40] holds.

Theorem 6 (GNN Transferability): Let $\mathbf{G}_{n_1}$ and $\mathbf{G}_{n_2}$, and $\mathbf{x}_{n_1}$ and $\mathbf{x}_{n_2}$, be graphs and graph signals obtained from the graphon $\mathbf{W}$ and the graphon signal X, as in (47), with $n_1 \neq n_2$. Consider the L-layer GNNs given by $\Phi(\mathbf{x}_{n_1}; \mathbf{H}, \mathbf{S}_{n_1})$ and $\Phi(\mathbf{x}_{n_2}; \mathbf{H}, \mathbf{S}_{n_2})$, where $F_\ell = 1$ for $1 \leq \ell \leq L$. Let the graph convolutions $h(\lambda)$ [see (34)] be such that $h(\lambda)$ is constant for $|\lambda| < c$. Then, under Assumptions 2–5, it holds

$$\begin{aligned}\|Y_{n_1} - Y_{n_2}\|_{L_2} &\\ \leq L\sqrt{A_1}&\left(A_2 + \frac{\pi B_c}{\delta_c}\right)\left({n_1}^{-\frac{1}{2}} + {n_2}^{-\frac{1}{2}}\right)\|X\|_{L_2}\\ &+ \frac{A_3}{\sqrt{3}}\left(n_1^{-\frac{1}{2}} + n_2^{-\frac{1}{2}}\right)\end{aligned}$$

where $Y_{n_j} = \Phi(X_{n_j}; \mathbf{H}, \mathbf{W}_{n_j})$ is the WNN induced by $\mathbf{y}_{n_j} = \Phi(\mathbf{x}_{n_j}; \mathbf{H}, \mathbf{S}_{n_j})$ [see (49)], $B_c = \max\{B_{n_1\ c}, B_{n_2\ c}\}$ [see Definition 5], and $\delta_c = \min\{\delta_{n_1\ c}, \delta_{n_2\ c}\}$ [see Definition 6].

Theorem 6 proves that GNNs are transferable between graphs of different sizes belonging to the same graphon family. This has two important implications. If the GNN hyperparameters are chosen carefully, the GNN can be transferred from the graph on which it was trained to another graph with an error bound inversely proportional to the sizes of both graphs. In scenarios where the same task has to be replicated on different graphs, for example, operating the same type of sensor network on multiple plants, this is the key because it avoids retraining the GNN. This result also implies that GNNs, such as graph filters, are scalable. They can be trained on smaller graphs than the graphs on which they are deployed (and vice versa) and are robust to increases in the graph size.

The approximation error is given by the transferability constant $LF^{L-1}\sqrt{A_1}(A_2 + \pi B_c/\delta_c)({n_1}^{-0.5} + {n_2}^{-0.5})$ and the fixed error term $A_3(n_1^{-0.5} + {n_2}^{-0.5})/\sqrt{3}$, both of which decrease asymptotically with n_1 and n_2. The fixed error term measures how different the graph signals $\mathbf{x}_{n_1}$ and $\mathbf{x}_{n_2}$ are from the graphon signal X; therefore, its contribution is small. The transferability constant, on the other hand, is determined by the graphon variability A_1, the number of layers L, and the convolutional filter parameters A_2, B_c, and δ_c. Except for A_1, all of these can be tuned. In order to have an asymptotic bound for $n_2 > n_1$, the number of eigenvalues in the band $[c, 1]$ must satisfy $B_c < \sqrt{n_1}$ [see Fig. 13]. This restriction is necessary to avoid mismatching the filter response for small eigenvalues of $\mathbf{G}_{n_1}$ and $\mathbf{G}_{n_2}$, which becomes harder to discriminate as they accumulate around zero [see Fig. 12]. As long as this condition is satisfied, the bound converges asymptotically because, as $n_1, n_2 \to \infty$, δ_c converges to the minimum eigengap of the graphon in the passing band.

The transferability bound in Theorem 6, thus, reflects a similar tradeoff between transferability and discriminability to that observed for graph filters. However, in the case of GNNs, this is partially overcome by the addition of nonlinearities. Nonlinearities act as rectifiers that *scatter* some spectral components associated with small λ around the middle range of the spectrum. This makes for an interesting parallel with the role of nonlinearities in stability, which depends on the components associated with large eigenvalues being scattered around the lower range of the spectrum instead.

VI. DECENTRALIZED COLLABORATIVE SYSTEMS

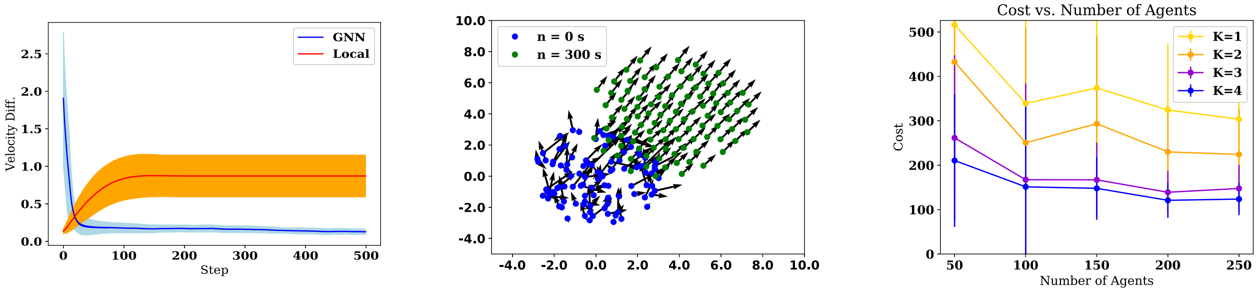
GNNs have been applied with success to learn decentralized control policies [7], [9]. Consider then a team of n agents that endeavor to accomplish a shared goal. Each agent has access to local states $\mathbf{x}_i$ and has to produce local control actions $\mathbf{a}_i$. Agent proximity determines the ability to exchange information between pairs of agents and results in access to delayed information about the state of the system. If agents i and j are separated by k communication hops, they know about their respective states with a delay of k time units. We capture this limitation with the definition of the information history of agent i:

$$\mathcal{X}_i(t) = \bigcup_{k=0}^{K-1} \left\{\mathbf{x}_j(t-k) : j \in \mathcal{N}_i^k(t)\right\}. \tag{51}$$

As per (51), agent i has access to its current state $\mathbf{x}_i(t)$ but only knows the states of k-hop neighbors at time $t - k$. A decentralized controller is one in which actions $\mathbf{a}_i(t)$ are functions of the history $\mathcal{X}_i(t)$. It is notable that the graph filters in (27) can be modified to have this property. Doing so requires that we rewrite (27) in terms of a diffusion sequence that takes time delays into consideration. Thus, replace $\mathbf{Z}_{lk}$ in (27) by $\mathbf{Z}_{lk}(t)$ defined as

$$\mathbf{Z}_{lk}(t) = \mathbf{S}\mathbf{Z}_{l,k-1}(t-1), \quad \text{with } \mathbf{Z}_{l0}(t) = \mathbf{X}_l(t). \tag{52}$$

This is the same as (26) except for the use of time delays to respect the information structure described by (51).



(a) (b) (c)

Fig. 14. ***GNN maintains a cohesive flock, while the local controller allows the flock to scatter. (a) Average difference in velocities. Local stands for K = 0. (b) Flock positions using the GNN. (c) Cost versus the number of agents.***

GNNs have proved successful in learning policies for flocking [7] and collaborative navigation [9]. We describe here some flocking results from [7]. In this scenario, we are given a team of n agents with random initial positions and velocities. The goal is for agents to form a cohesive flock, in which: 1) they all move with the same velocity and 2) there are no collisions between agents. To solve this problem, we consider local states $\mathbf{x}_i(t) \in \mathbb{R}^6$ with components

$$\mathbf{x}_i^T(t) = \left[\sum_{j\in\mathcal{N}_i} \mathbf{v}_{ij}(t); \sum_{j\in\mathcal{N}_i(t)} \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}(t)\|^4}; \sum_{j\in\mathcal{N}_i} \frac{\mathbf{r}_{ij}(t)}{\|\mathbf{r}_{ij}(t)\|^2}\right]. \tag{53}$$

In (53), $\mathbf{r}_{ij}(t)$ and $\mathbf{v}_{ij}(t)$ denote the positions and velocities of agent j measured relative to the position and velocity of agent i, respectively. The neighborhood $\mathcal{N}_i$ is made up of nodes j for which the distance $\|\mathbf{r}_{ij}\| \leq R$. The distance R represents a communication and sensing radius. The components of the state in (53) are somewhat arbitrary. They are motivated by their use in a benchmark decentralized controller [70].

It is important to observe that an optimal centralized controller is trivial as we can just order all the agents to move in the same direction. The optimal decentralized controller is, however, unknown. We, therefore, choose to train a decentralized GNN to mimic the centralized controller while respecting the information structure in (51). While perfect mimicry is not attained, we do observe improvement relative to existing decentralized controllers. This is illustrated in Fig. 14(a) and (b) where we show the velocities for a swarm that is controlled with a GNN and a swarm that is controlled with the decentralized controller in [70]. A more comprehensive evaluation is shown in Fig. 14(c) where we illustrate the cost that is attained by different GNN architectures as we vary the flock size. The ability to attain a small cost for large swarms is worth emphasizing.

Since we are training to mimic a centralized controller, the training of the GNN is an offline process. This fact implies that the networks that are observed during training and the networks that are observed during execution are different. This is not expected to be an issue because of the stability and transferability results of Sections IV and V. The numerical results in Fig. 14 corroborate that this is true.

VII. WIRELESS COMMUNICATION NETWORKS

GNNs have also been applied with success to learn optimal resource allocations in wireless communication networks [58]. Consider an *ad hoc* wireless network with n transmitter and receiver pairs indexed by $i \in \{1, n\}$. Wireless link states are represented with fading coefficients $s_{ij} \in \mathbb{R}_+$, which denotes the fading state between a transmitter i and a receiver j. The fading channel s_{ii} connects transmitter i to its intended receiver j. The fading channel s_{ij} with $j \neq i$ links i to other receivers on which the transmission of i manifests as interference. All channels are arranged in the matrix $\mathbf{S} \in \mathbb{R}^{m\times m}$. The goal is to map fading state observations $\mathbf{S}$ to power allocations $\mathbf{p} := [p_1; \ldots; p_m] = \mathbf{p}(\mathbf{S})$. The combination of channel realizations $\mathbf{S}$ and power allocations $\mathbf{p}(\mathbf{S})$ determines the communication rate between each transmitter–receiver pair. For instance, if using capacity achieving codes without interference cancellation, rates are determined by the

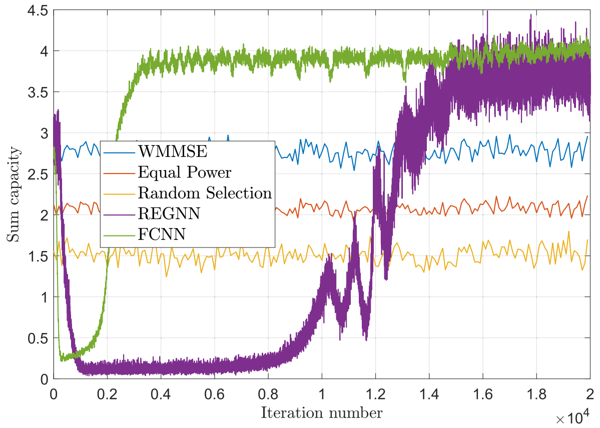


Fig. 15. ***Performance of GNN during training for m = 20 pairs, in comparison with FCNN and three heuristic baselines: WMMSE [71], equal power division across all users, and across a random subset of users.***

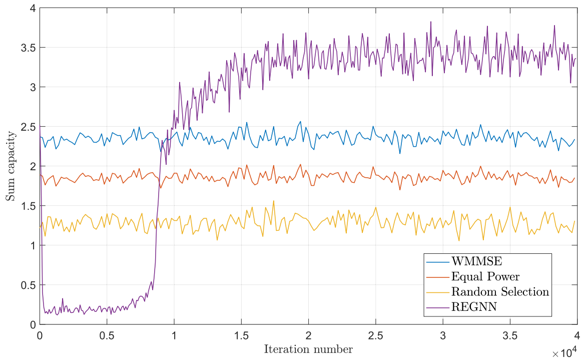


Fig. 16. ***Performance of GNN during training for $m = 50$ pairs, in comparison with three heuristic baselines: WMMSE [71], equal power division across all users, and across a random subset of users.***

function

$$f_i(\mathbf{p};\mathbf{S}) := \log\left(1 + \frac{s_{ii}p_i(\mathbf{S})}{1+\sum_{j\neq i} s_{ji}p_j(\mathbf{S})}\right). \tag{54}$$

The expression in (54) represents an instantaneous performance metric. It is customary to focus on the long-term performance given by the expectation $\mathbb{E}[f_i(\mathbf{p};\mathbf{S})]$ over realizations of the fading channels $\mathbf{S}$. A particular problem of interest is the maximization of the expected sum rate, which leads to the optimal power allocation being given by

$$\mathbf{p}^*(\mathbf{S}) \;=\; \operatorname{argmax} \sum_{i=1}^{n} \mathbb{E}[f_i(\mathbf{p};\mathbf{S})]. \tag{55}$$

The problem in (55) is a statistical risk minimization problem of the form in (II). We advocate its solution with a GNN and, therefore, choose to parameterize the power allocation as a $\mathbf{p}(\mathbf{S}) = \Phi(\mathbf{x};\mathcal{H},\mathbf{S})$. The important observation to make is that, in (55), we want to find a power allocation $\mathbf{p}(\mathbf{S})$ associated with each fading realization $\mathbf{S}$. Thus, we are reinterpreting the shift operator $\mathbf{S}$ as an input to the GNN. To emphasize this fact, we say that the parameterization is a random-edge (RE)GNN. There is also no input $\mathbf{x}$ in (55). We can, therefore, set $\mathbf{x} = \mathbf{1}$ in the GNN parameterization.

Figs. 15 and 16 show training curves for the solution of (55) with an REGNN parameterization [58]. For comparison, training curves for an FCNN are also shown along with heuristics [71]. Fig. 15 considers 20 communicating pairs. It is notable that both the REGNN and the FCNN outperform existing heuristics and attain similar performance. The advantage of the REGNN is that it utilizes a smaller number of parameters. In Fig. 16, we consider 50 communicating pairs. The REGNN still outperforms standard heuristics. Missing from this picture is a curve for an FCNN. This is because it fails to train in a network of this size.

The formulation in (54) and (55) can be generalized to different rate functions, and it can be modified to incorporate constraints and network state representations. We refer the interested reader to [58].

VIII. CONCLUSION

GNNs are becoming the tool of choice for the processing of signals supported on graphs. In this article, we have shown that GNNs are minor variations of graph convolutional filters. They differ in the incorporation of pointwise nonlinear functions and the addition of multiple layers. Being minor variations of graph filters, the good empirical performance of GNNs is expected: we have ample evidence supporting the usefulness of graph filters. What is unexpected is the appearance of significant gains for what is such a minor variation. In this article, we attempted to explain this phenomenon with a perturbation stability analysis, showing that pointwise nonlinearities make it possible to discriminate signals while retaining robustness with respect to perturbations of the graph.

We further introduced graphon filters and graphon neural networks so as to understand the limit behavior of GNNs. This analysis uncovers the ability to transfer a GNN across graphs with different numbers of nodes. As in the case of our stability analysis, we discovered that GNNs exhibit more robust transferability than linear graph filters.

In both domains, there remains much to be done. For instance, our stability analysis has much to say about the perturbation of eigenvalues of a graph shift operator but little to say about the perturbation of its eigenvectors. There are also other ways of defining graph limits that are not graphons and several other GNN architectures whose fundamental properties have not been studied. We hope that this contribution can spark interest in understanding the fundamental properties of GNNs. ■

REFERENCES

[1] T. Joachims, "A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization," in *Proc. 14th Int. Conf. Mach. Learn.*, Nashville, TN, USA, Jul. 1997, pp. 143–151.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. 1st Int. Conf. Learn. Represent.* Scottsdale, AZ, USA: Assoc. Comput. Linguistics, May 2013, pp. 1–12.

[3] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. 30th Conf. Neural Inf. Process. Syst.* Barcelona, Spain: Neural Inf. Process. Found., Dec. 2016, pp. 3844–3858.

[4] W. Huang, A. G. Marques, and A. R. Ribeiro, "Rating prediction via graph signal processing," *IEEE Trans. Signal Process.*, vol. 66, no. 19, pp. 5066–5081, Oct. 2018.

[5] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for Web-scale recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. London, U.K.: Assoc. Comput. Machinery, Jul. 2018, pp. 974–983.

[6] F. Monti, M. M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Proc. 31st Conf. Neural Inf. Process. Syst.* Long Beach, CA, USA: Neural Inf. Process. Found., Dec. 2017, pp. 3697–3707.

[7] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Proc. Conf. Robot Learn.*, vol. 100. Osaka, Japan: Proc. Mach. Learn. Res., Oct./Nov. 2019, pp. 1–12.

[8] G. Sartoretti *et al.*, "PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2378–2385, Jul. 2019.

[9] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst.* Las Vegas, NV, USA, Oct. 2020, pp. 1–8.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (The Adaptive Computation and Machine Learning Series). Cambridge, MA, USA: MIT Press, 2016.

[11] 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.

[12] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Trans. Signal Process.*, vol. 65, no. 15, pp. 4117–4131, Aug. 2017.

[13] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *Proc. 2nd Int. Conf. Learn. Represent.* Banff, AB, Canada: Assoc. Comput. Linguistics, Apr. 2014, pp. 1–14.

[14] 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.

[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*. Toulon, France: Assoc. Comput. Linguistics, Apr. 2017, pp. 1–14.

[16] E. Isufi, F. Gama, and A. Ribeiro, "EdgeNets: Edge varying graph neural networks," 2020, *arXiv:2001.07620*. [Online]. Available: http://arxiv.org/abs/2001.07620

[17] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, p. 19, Jan. 2016.

[18] S. Mallat, "Group invariant scattering," *Commun. Pure Appl. Math.*, vol. 65, no. 10, pp. 1331–1398, Oct. 2012.

[19] M. Assran and M. Rabbat, "On the convergence of Nesterov's accelerated gradient method in stochastic settings," in *Proc. 37th Int. Conf. Mach. Learn.*, Vienna, Austria, Jul. 2020, pp. 1–20.

[20] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.* San Diego, CA, USA: Assoc. Comput. Linguistics, May 2015, pp. 1–15.

[21] 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.

[22] A. Ortega, P. Frossard, J. Kovacevic, 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.

[23] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Autoregressive moving average graph filtering," *IEEE Trans. Signal Process.*, vol. 65, no. 2, pp. 274–288, Jan. 2017.

[24] N. Tremblay, P. Gonçalves, and P. Borgnat, "Design of graph filters and filterbanks," in *Cooperative and Graph Signal Processing*. Amsterdam, The Netherlands: Elsevier, 2018, pp. 299–324.

[25] J. Du, S. Zhang, G. Wu, J. M. F. Moura, and S. Kar, "Topology adaptive graph convolutional networks," 2017, *arXiv:1710.10370*. [Online]. Available: http://arxiv.org/abs/1710.10370

[26] J. Du, J. Shi, S. Kar, and J. M. F. Moura, "On graph convolution for graph CNNs," in *Proc. IEEE Data Sci. Workshop (DSW)*, Lausanne, Switzerland, Jun. 2018, pp. 239–243.

[27] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. 7th Int. Conf. Learn. Represent.* New Orleans, LA, USA: Assoc. Comput. Linguistics, May 2019, pp. 1–17.

[28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada: Assoc. Comput. Linguistics, Feb. 2018, pp. 1–12.

[29] L. Ruiz, F. Gama, A. G. Marques, and A. Ribeiro, "Invariance-preserving localized activation functions for graph neural networks," *IEEE Trans. Signal Process.*, vol. 68, pp. 127–141, Jan. 2020.

[30] L. Ruiz, F. Gama, and A. Ribeiro, "Gated graph recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 68, pp. 6303–6318, 2020.

[31] V. N. Ioannidis, A. G. Marques, and G. B. Giannakis, "A recurrent graph neural network for multi-relational data," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Brighton, U.K., May 2019, pp. 8157–8161.

[32] X. Bresson and T. Laurent, "Residual gated graph ConvNets," 2017, *arXiv:1711.07553*. [Online]. Available: http://arxiv.org/abs/1711.07553

[33] M. Coutino, E. Isufi, and G. Leus, "Advances in distributed graph filtering," *IEEE Trans. Signal Process.*, vol. 67, no. 9, pp. 2320–2333, May 2019.

[34] G. Boukli Hacene, C. Lassance, V. Gripon, M. Courbariaux, and Y. Bengio, "Attention based pruning for shift networks," 2019, *arXiv:1905.12300*. [Online]. Available: http://arxiv.org/abs/1905.12300

[35] G. Wang, R. Ying, J. Huang, and J. Leskovec, "Direct multi-hop attention based graph neural network," 2020, *arXiv:2009.14332*. [Online]. Available: http://arxiv.org/abs/2009.14332

[36] L. Gong and Q. Cheng, "Exploiting edge features for graph neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9211–9219.

[37] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. 6th Int. Conf. Learn. Represent.* Vancouver, BC, Canada: Assoc. Comput. Linguistics, Apr./May 2018, pp. 1–16.

[38] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *Proc. 32nd Conf. Neural Inf. Process. Syst.* Montreal, QC, Canada: Neural Inform. Process. Foundation, Dec. 2018, pp. 362–373.

[39] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *IEEE Trans. Signal Process.*, vol. 68, pp. 5680–5695, Sep. 2020.

[40] L. Ruiz, L. F. O. Chamon, and A. Ribeiro, "Graphon neural networks and the transferability of graph neural networks," in *Proc. 34th Conf. Neural Inf. Process. Syst.* Vancouver, BC, Canada: Neural Inform. Process. Foundation, Dec. 2020, pp. 1–11.

[41] D. Zou and G. Lerman, "Graph convolutional neural networks via scattering," *Appl. Comput. Harmon. Anal.*, vol. 49, no. 3, pp. 1046–1074, Nov. 2020, doi: 10.1016/j.acha.2019.06.003.

[42] F. Gama, A. Ribeiro, and J. Bruna, "Diffusion scattering transforms on graphs," in *Proc. 7th Int. Conf. Learn. Represent.* New Orleans, LA, USA: Assoc. Comput. Linguistics, May 2019, pp. 1–12.

[43] Z. Chen, S. Villar, L. Chen, and J. Bruna, "On the equivalence between graph isomorphism testing and function approximation with GNNs," in *Proc. 33rd Conf. Neural Inf. Process. Syst.* Vancouver, BC, Canada: Neural Inf. Process. Found., Dec. 2019, pp. 1–19.

[44] C. Vignac, A. Loukas, and P. Frossard, "Building powerful and equivariant graph neural networks with message-passing," in *Proc. 34th Conf. Neural Inf. Process. Syst.* Vancouver, BC, Canada: Neural Inf. Process. Found., Dec. 2020, pp. 1–21.

[45] L. Lovász, *Large Networks and Graph Limits*, vol. 60. Providence, RI, USA: American Mathematical Society, 2012.

[46] C. Borgs, J. Chayes, L. Lovász, V. Sós, and K. Vesztergombi, "Convergent sequences of dense graphs II. Multiway cuts and statistical physics," *Ann. Math.*, vol. 176, no. 1, pp. 151–219, Jul. 2012.

[47] P. J. Wolfe and S. C. Olhede, "Nonparametric graphon estimation," 2013, *arXiv:1309.5936*. [Online]. Available: http://arxiv.org/abs/1309.5936

[48] M. Avella-Medina, F. Parise, M. T. Schaub, and S. Segarra, "Centrality measures for graphons: Accounting for uncertainty in networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 520–537, Jan. 2020.

[49] F. Parise and A. Ozdaglar, "Graphon games," in *Proc. ACM Conf. Econ. Comput.*, Jun. 2019, pp. 457–458.

[50] L. Ruiz, L. F. O. Chamon, and A. Ribeiro, "Graphon signal processing," 2020, *arXiv:2003.05030*. [Online]. Available: http://arxiv.org/abs/2003.05030

[51] M. W. Morency and G. Leus, "Signal processing on kernel-based random graphs," in *Proc. 25th Eur. Signal Process. Conf. (EUSIPCO)*, Aug. 2017, pp. 365–369.

[52] R. Levie, W. Huang, L. Bucci, M. M. Bronstein, and G. Kutyniok, "Transferability of spectral graph convolutional neural networks," 2019, *arXiv:1907.12972*. [Online]. Available: http://arxiv.org/abs/1907.12972

[53] S. I. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. Lee, B. Glocker, and D. Rueckert, "Distance metric learning using graph convolutional networks: Application to functional brain networks," in *Proc. Int. Conf. Med. Image Comput. Comput. Assist. Intervent.* Cham, Switzerland: Springer, 2017, pp. 469–477.

[54] D. K. Duvenaud *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2224–2232.

[55] F. Scarselli, S. Liang Yong, M. Gori, M. Hagenbuchner, A. C. Tsoi, and M. Maggini, "Graph neural networks for ranking Web pages," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. (WI)*, 2005, pp. 666–672.

[56] C. Vignac, G. Ortiz-Jimenez, and P. Frossard, "On the choice of graph neural network architectures," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 8489–8493.

[57] D. Owerko, F. Gama, and A. Ribeiro, "Optimal power flow using graph neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Barcelona, Spain, May 2020, pp. 5930–5934.

[58] M. Eisen and A. Ribeiro, "Optimal wireless resource allocation with random edge graph neural networks," *IEEE Trans. Signal Process.*, vol. 68, pp. 2977–2991, Apr. 2020.

[59] M. Cheung, J. Shi, O. Wright, L. Y. Jiang, X. Liu, and J. M. F. Moura, "Graph signal processing and deep learning: Convolution, pooling, and topology," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 139–149, Nov. 2020.

[60] B. Pasdeloup, V. Gripon, R. Alami, and M. G. Rabbat, "Uncertainty principle on graphs," in *Vertex-Frequency Analysis of Graph Signals*. Cham, Switzerland: Springer, 2019, pp. 317–340.

[61] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs: Frequency analysis," *IEEE Trans. Signal Process.*, vol. 62, no. 12, pp. 3042–3054, Jun. 2014.

[62] B. Ricaud, P. Borgnat, N. Tremblay, P. Gonçalves, and P. Vandergheynst, "Fourier could be a data scientist: From graph Fourier transform to signal processing on graphs," *Comp. Rendus Phys.*, vol. 20, no. 5, pp. 474–488, Jul. 2019.

[63] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Upper Saddle River, NJ, USA: Pearson, 2010.

[64] I. Daubechies, *Ten Lectures on Wavelets* (CBMS-NSF Regional Conference Series in Applied Mathematics), vol. 61. Philadelphia, PA, USA: SIAM, 1992.

[65] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmon. Anal.*, vol. 30, no. 2, pp. 129–150, Mar. 2011.

[66] 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.

[67] J. Wishart, "The generalised product moment distribution in samples from a normal multivariate population," *Biometrika*, vol. 20A, nos. 1–2, pp. 32–52, 1928.

[68] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). New York, NY, USA: Springer, 2006.

[69] P. D. Lax, *Functional Analysis*. Hoboken, NJ, USA: Wiley, 2002.

[70] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable flocking of mobile agents part II: Dynamic topology," in *Proc. 42nd IEEE Conf. Decis. Control*, vol. 2, Dec. 2003, pp. 2016–2021.

[71] Q. Shi, M. Razaviyayn, Z.-Q. Luo, and C. He, "An iteratively weighted MMSE approach to distributed sum-utility maximization for a MIMO interfering broadcast channel," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2011, pp. 3060–3063.