Sheaf Neural Networks

Jakob Hansen*
Department of Mathematics
The Ohio State University
hansen.612@osu.edu

Thomas Gebhart*
Department of Computer Science
University of Minnesota
gebhart@umn.edu

Abstract

We present a generalization of graph convolutional networks by generalizing the diffusion operation underlying this class of graph neural networks. These *sheaf neural networks* are based on the *sheaf Laplacian*, a generalization of the graph Laplacian that encodes additional relational structure parameterized by the underlying graph. The sheaf Laplacian and associated matrices provide an extended version of the diffusion operation in graph convolutional networks, providing a proper generalization for domains where relations between nodes are non-constant, asymmetric, and varying in dimension. We show that the resulting sheaf neural networks can outperform graph convolutional networks in domains where relations between nodes are asymmetric and signed.

1 Introduction

Graph neural networks are a class of generalized neural network architectures that take as input relational data and learn to classify the input graph or its nodes. Because this relational data lacks a constant local Euclidean structure, the definition of a local convolution-type operator through which weight sharing may be achieved is a non-trivial architectural challenge. A natural approach, inspired by the theory of graph signal processing [8] is to define convolution via the graph Laplacian or adjacency matrices [1, 3, 7] so that the layer-wise convolution operation acts as neighborhood averaging followed by propagation of the updated signal, akin to message-passing diffusion with an additional element-wise non-linearity [11]. The association with convolution comes from taking the eigenvectors of the Laplacian or adjacency matrix as a Fourier basis for signals on the graph; convolution is then defined as multiplication of signals in this spectral domain. Actually computing the spectral coefficients is computationally intensive, so it is common to instead parameterize convolutional filters by polynomials in the adjacency or Laplacian operators. Indeed, using a polynomial of degree 1 works well in many situations [7] and provides a computationally efficient architecture. These convolution operations, broadly construed, define a class of graph neural networks which we refer to as Graph Convolutional Networks (GCN).

We introduce a generalization of this GCN framework by leveraging a natural extension of the graph Laplacian called the sheaf Laplacian [5]. Sheaf Laplacians form a class of local linear operators on a graph that respect the topological and algebraic structure of data associated to nodes. This generalization allows more complex relationships between nodes to be expressed while maintaining a consistent notion of local averaging and diffusion. After introducing cellular sheaves, sheaf Laplacians, and sheaf diffusion operators, we define sheaf neural networks and show that they are a useful generalization of GCNs in domains where the relational interaction between nodes across edges is non-constant. We validate this claim by comparing the performance of sheaf neural networks to GCNs on synthetic, semisupervised classification problems defined over signed graphs and find that sheaf neural networks significantly outperform the Kipf-Welling GCNs in this domain.

Topological Data Analysis and Beyond Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

2 Cellular sheaves

A *cellular sheaf* is an algebraic-topological structure associated with a graph that attaches spaces of data to nodes and edges. To be precise, a cellular sheaf $\mathcal{F}$ on an undirected graph G is given by specifying

- a vector space $\mathcal{F}(v)$ for each vertex v of G
- a vector space $\mathcal{F}(e)$ for each edge e of G, and
- a linear map $\mathcal{F}_{v \trianglelefteq e} : \mathcal{F}(v) \to \mathcal{F}(e)$ for each incident vertex-edge pair $v \trianglelefteq e$ of G.

The sheaf structure assigns spaces of data to vertices and edges, and specifies consistency constraints for this data. For an edge e between vertices u and v, we say that a choice of data $x_v \in \mathcal{F}(v)$, $x_u \in \mathcal{F}(u)$ is *consistent* over e if $\mathcal{F}_{v \trianglelefteq e} x_v = \mathcal{F}_{u \trianglelefteq e} x_u$. The space of data associated with all vertices of G is denoted $C^0(G;\mathcal{F})$ and called the space of 0-cochains valued in $\mathcal{F}$. One thinks of $C^0(G;\mathcal{F})$ as a space of *signals* on the vertices of G, where the value of a signal at a vertex v lives in the vector space $\mathcal{F}(v)$. Each edge of G imposes a constraint on $C^0(G;\mathcal{F})$ by restricting the space associated with its two incident vertices. The subspace of $C^0(G;\mathcal{F})$ satisfying all these constraints is the space of *global sections* of $\mathcal{F}$, and is denoted $H^0(G;\mathcal{F})$.

There is likewise a space of signals associated with edges, denoted $C^1(G;\mathcal{F})$. The space of global sections $H^0(G;\mathcal{F})$ is the kernel of a linear map $\delta : C^0(G;\mathcal{F}) \to C^1(G;\mathcal{F})$. This map is called the *coboundary*, and, given an arbitrary choice of orientation on the edges of the graph, may be computed by

$$(\delta x)_e = \mathcal{F}_{v \trianglelefteq e} x_v - \mathcal{F}_{u \trianglelefteq e} x_u$$

for each oriented edge $e = u \to v$. Clearly, if $x \in \ker \delta$, then $\mathcal{F}_{v \trianglelefteq e} x_v = \mathcal{F}_{u \trianglelefteq e} x_u$ for every edge $e = u \sim v$. From the coboundary operator we may construct the *sheaf Laplacian* $L_{\mathcal{F}} = \delta^T \delta$, which is a positive semidefinite linear operator on $C^0(G;\mathcal{F})$ with kernel $H^0(G;\mathcal{F})$, and does not depend on the orientation chosen for the edges of G.

A cellular sheaf operates as an extension of the structure of a graph. Rather than simply recording connections between nodes, it specifies relationships between data associated with those nodes. Standard graph-theoretic constructions like Laplacians and diffusion operators implicitly work with the *constant sheaf* on a graph: the sheaf $\underline{\mathbb{R}}$ with all stalks $\mathbb{R}$ and all restriction maps the identity. This is a very simple sort of relationship between nodes, and can be greatly generalized in the sheaf setting. For instance, a sheaf can easily represent a signed graph by changing the sign of one restriction map of the constant sheaf for each negatively signed edge. Even more general relationships between nodes can be expressed, especially as stalks increase in dimension, resulting in such operators as connection Laplacians [9] and matrix-weighted Laplacians [10]. In this paper we will simplify the constructions by only considering sheaves with constant-dimensional vertex stalks $\mathbb{R}^k$. More information on cellular sheaves and their Laplacians may be found in [2, 5], including extensions to higher-dimensional base spaces and additional topological context.

2.1 Sheaf diffusion operators

By an r-step *local operator* associated to $\mathcal{F}$ we mean any linear operator $D_{\mathcal{F}}$ on $C^0(G;\mathcal{F})$ which is local with respect to G, in the sense that $(D_{\mathcal{F}} x)_v$ depends only on x_u for nodes u in the r-step neighborhood of v. We will suggestively call such an operator a *sheaf diffusion operator* if it has nice properties with respect to the algebraic structure of $\mathcal{F}$—for instance, if sections of $\mathcal{F}$ form an eigenspace of $D_{\mathcal{F}}$. One such operator is the sheaf Laplacian $L_{\mathcal{F}} = \delta^T \delta$. This is a 1-step diffusion operator whose zero eigenspace consists of sections of $\mathcal{F}$. For an appropriately chosen α, the operator $H^{\alpha}_{\mathcal{F}} = I - \alpha L_{\mathcal{F}}$ will have 2-norm 1 and have $H^0(G;\mathcal{F})$ as the eigenspace corresponding to the eigenvalue 1. There is also a normalized form of the sheaf Laplacian, $\tilde{L}_{\mathcal{F}} = D^{-1/2} L_{\mathcal{F}} D^{-1/2}$, where D is the block diagonal of $L_{\mathcal{F}}$. It amounts to the Laplacian for a version of $\mathcal{F}$ with reparameterized stalks, and its eigenvalues lie between 0 and 2. This means that no scaling factor is necessary to construct a stable diffusion operator $\tilde{H}_{\mathcal{F}} = I - \tilde{L}_{\mathcal{F}}$. Diffusion operators depending on larger neighborhoods may be constructed from powers of these operators. For any r, $(L_{\mathcal{F}})^r$ and $(H^{\alpha}_{\mathcal{F}})^r$ are r-step sheaf diffusion operators.

From the standpoint of graph signal processing [8], we can consider a sheaf diffusion operator $D_{\mathcal{F}}$ as a shift operator generating convolution-like filters for signals in $C^0(G;\mathcal{F})$. We can define

the convolution of two signals $x, y \in C^0(G;\mathcal{F})$ by taking an eigendecomposition $D_{\mathcal{F}} = S\Lambda S^{-1}$ and letting $x * y = S(S^{-1}x \circ S^{-1}y)$, i.e. by pointwise multiplication in the eigenbasis of $D_{\mathcal{F}}$. By standard algebraic arguments, for a fixed $y \in C^0(G;\mathcal{F})$ one can find a polynomial $P = a_0 I + a_1 D_{\mathcal{F}} + \cdots + a_N D_{\mathcal{F}}^N$ in $D_{\mathcal{F}}$ such that $x * y = Px$. Thus we can parameterize sheaf convolutions by polynomials in $D_{\mathcal{F}}$, without having to compute an eigendecomposition. This parameterization also gives natural bases for restricting the dimension of the space of convolutional filters. Indeed, in the context of graph convolutional networks, it is common to use a a degree-1 polynomial, since this significantly reduces the number of parameters to learn.

3 Sheaf neural networks

Local graph operators have been used to construct GCNs. These typically rely on applying one or more graph diffusion operators to the features on each layer and then taking a node-independent linear combination of features. A *sheaf neural network* is an extension of this framework.

Suppose that each node has N_{feat} k-dimensional features. We write the array of all features for all nodes as an $N_v k \times N_{\text{feat}}$ matrix X. A single forward diffusion step applies a sheaf diffusion operator $D_{\mathcal{F}}$ for some sheaf $\mathcal{F}$ with k-dimensional stalks to the feature matrix X; that is, it calculates $D_{\mathcal{F}}X$, producing a new set of N_{feat} k-dimensional features for each node. These features are neighborhood mixtures of the previous features, computed in a way that is consistent with the algebraic structure of the sheaf. To construct a neural network layer, we also apply a linear operator to each nodewise feature vector. When $k = 1$ this is realized by a right multiplication XA for some matrix A with N_{feat} rows. For $k > 1$ one must also choose some $k \times k$ matrix B and multiply it on the left to each block row of X. That is, the total operation on X is $(I \otimes B)XA$, where $I \otimes B$ is the Kronecker product of B with the $N_v \times N_v$ identity matrix; this amounts to a block diagonal matrix with copies of B on the diagonal. After these linear transformations, we apply a stalkwise nonlinearity $\rho : \mathbb{R}^k \to \mathbb{R}^k$.

Combining these, a sheaf neural network layer for a sheaf $\mathcal{F}$ on G has hyperparameters $N_{\text{feat}}^{\text{in}}$, $N_{\text{feat}}^{\text{out}}$, and $D_{\mathcal{F}}$, a diffusion operator associated with $\mathcal{F}$, together with a stalkwise nonlinearity $\rho : \mathbb{R}^k \to \mathbb{R}^k$. Its learnable parameters are the $N_{\text{feat}}^{\text{in}} \times N_{\text{feat}}^{\text{out}}$ matrix A and the $k \times k$ matrix B. The action of this layer $\text{SheafConv}(A, B)$ on a matrix X of nodewise features is

$$\text{SheafConv}(A, B)(X) = \rho\left(D_{\mathcal{F}}(I \otimes B)XA\right).$$

We can use multiple diffusion operators in parallel by taking a concatenation or learnable linear combination of sheaf neural network layers (without the nonlinearity ρ). This might be useful for a task where we expect some behavior to be driven purely by connectivity, while other behavior depends on the relationships between nodes, and can also be used to combine powers of a diffusion operator to parameterize higher-order sheaf convolutional filters.

4 Semisupervised classification

One problem for which graph-based neural networks are frequently used is the semisupervised classification problem. In this problem, a network of objects is given, together with $N_{\text{feat}}^{\text{in}}$ nodewise features and N_{class} class labels associated with a (typically small) subset of the nodes. The goal is to impute class labels for the remaining nodes. For our sheaf-based approach to meaningfully differ from a simpler graph neural network, we need an appropriate sheaf from which to construct a diffusion operator. Unfortunately, many popular graph classification datasets do not admit obvious sheaf Laplacian operators that are meaningfully different from the standard graph Laplacian. Due to the lack of appropriate benchmark datasets, we illustrate the potential for sheaf-based neural networks using a synthetic family of semisupervised node classification problems over signed graphs. This allows us to isolate the contribution of a correctly-chosen sheaf diffusion operator in an appropriate setting.

To generate the synthetic node classification problem, we begin with a set of N_v nodes, and assign each node an intrinsic feature vector $x_v \in \mathbb{R}^{N_{\text{intrinsic}}}$, sampled from a standard normal distribution. We choose a classification vector $c \in \mathbb{R}^{N_{\text{intrinsic}}}$, and for each node v choose a class $C_v \in \{\pm 1\}$ by $C_v = \text{sign}(\langle c, x_v \rangle)$. We consider two sets of input features: one linear in the intrinsic features and one nonlinear. The linear features are a noisy random linear transformation of the intrinsic features: $x_v^{\text{in}} = Px_v + \epsilon_v$, where P is a random $N_{\text{feat}}^{\text{in}} \times N_{\text{intrinsic}}$ matrix with independent standard Gaussian

entries and ϵ is a noise term obtained by independent samples from a normal distribution with mean zero and variance σ^2_{feat}. The nonlinear feature regime extends the map producing the linear features to a random 2-layer fully connected neural network: $x_v^{\text{in}} = P_2\text{ReLU}(P_1 x_v + \epsilon_v)$, where P_1 and P_2 are random matrices with independent standard normal entries. We include this nonlinear regime to provide a stronger inferential challenge. It helps evaluate the potential performance of the SheafNN architecture for problems where the features and classes are not linearly related.

We generate a noisy graph from the intrinsic features by computing signed weights $w_{uv} = \langle x_u, x_v \rangle + \epsilon_{uv}$ and imputing a signed edge with weight w_{uv} when $|w_{uv}| > \tau$. Here ϵ_{uv} is drawn from a mean-zero Gaussian with variance σ_w^2. From this signed graph data, we construct a cellular sheaf $\mathcal{F}$ with all stalks $\mathbb{R}$, where the restriction maps $\mathcal{F}_{v \trianglelefteq e}$ and $\mathcal{F}_{u \trianglelefteq e}$ for an edge $e = u \sim v$ are multiplication by $\pm\sqrt{|w_{uv}|}$. The signs are determined by the parity of the edge: for a positively signed edge $\mathcal{F}_{v \trianglelefteq e}$ and $\mathcal{F}_{u \trianglelefteq e}$ have the same sign, and for a negatively signed edge they have the opposite sign. The diffusion operator used is then $D_{\mathcal{F}} = H_{\mathcal{F}}^{1/d_{\max}} = I - \frac{1}{d_{\max}} L_{\mathcal{F}}$, where $d_{\max}$ is the maximum degree of the underlying graph.

For both the linear and non-linear noise regimes, we initialize two diffusion-based sheaf neural networks. The first architecture is three layers with 32 hidden dimensions in each layer (SheafNN-32). The second architecture is four layers with 16 hidden dimensions in each layer (SheafNN-16). For comparison, we also initialize two traditional diffusion-based graph convolutional networks according to [7] with equivalent hidden dimensions and number of layers (GCN-32, GCN-16). All models have ReLU activations. For each model type, we instantiate five random networks of 5000 nodes and simulate noisy intrinsic data according to the procedure in the previous paragraph. We set $N_{\text{intrinsic}} = 25, \tau = 0.5, N_{\text{feat}}^{\text{in}} = 32$ and train each model according to this binary classification task, training on 75% of nodes and determining test performance on the rest. We train each model for 1000 epochs using Adam gradient optimization [6] with learning rate 0.001.

Figure 1 plots the performance of these two models in both the linear and nonlinear feature generation regimes. Clearly, the SheafNN variants nearly all outperform the GCN variants under a wide range of feature noise levels (σ^2_{feat}) and weight noise levels (σ_w^2). The GCN models appear to saturate in accuracy slightly above chance. This is expected as their underlying diffusion operation does not respect the signed nature of the graph. Increasing the weight noise level has the effect of decreasing performance and slowing training convergence. This is expected as increasing the noise across edge weights blurs the intrinsic similarity between nodes which has the effect of promoting false relations between nodes, masking true relations between intrinsically similar nodes, or flipping the intrinsic relationship altogether. Increasing feature noise uniformly dampens the maximal classification accuracy across all models as the underlying signal becomes distorted.

5 Discussion

The sheaf Laplacian is as a proper generalization of the graph Laplacian in defining diffusion operations for domains where relations between nodes are non-constant, asymmetric, and varying in dimension. This sheaf diffusion operator acts as a drop-in replacement for the graph diffusion operator and outperforms the standard graph convolutional model in semisupervised node classification over signed graphs. There are many avenues for future research related to sheaf neural networks.

As mentioned in Section 4, most standard graph datasets do not offer obvious sheaf structures to leverage, which makes applying sheaf neural networks a more difficult task. We view this fact as a reflection of the early stage of development of graph classification and processing. There are many relational processes in nature that are most naturally modeled using asymmetric, heterogeneous relations. As datasets related to these processes emerge, we expect sheaf neural networks to outperform related algorithms defined using adjacency-based diffusion operators. Another possible avenue of application is to learn the sheaf structure from the dataset, as in [4]. One exciting possibility is the combination of these approaches, where the sheaf structure is learned simultaneously with the solution to a classification or regression task.

Finally, we note that we have merely scratched the surface of cellular sheaf theory in the above sections. Numerous other ideas from cellular sheaf theory—sheaf morphisms, approximations, and pushforward/pullback operations—could offer even greater flexibility in analyzing graph datasets.

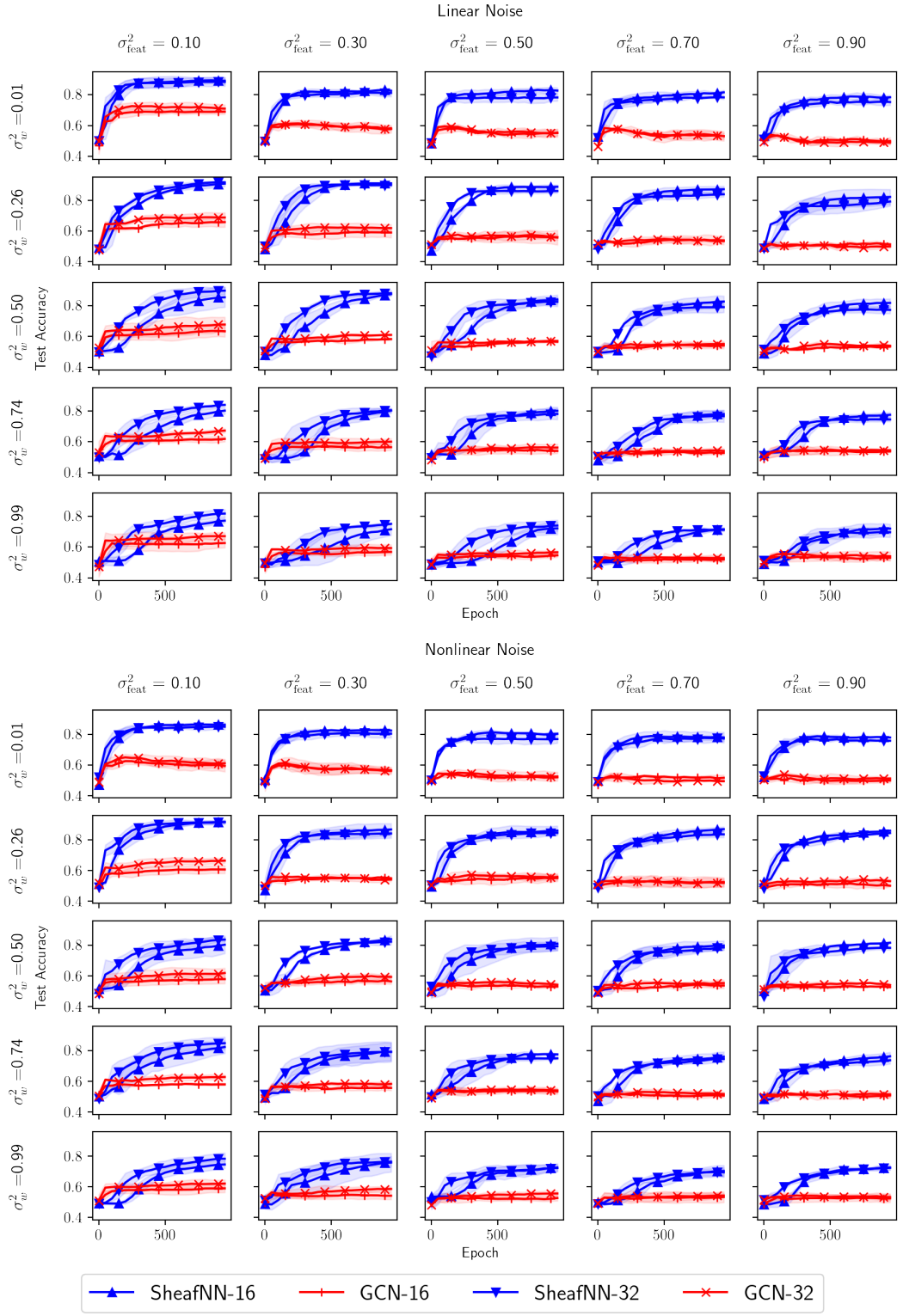

Figure 1: **Sheaf neural networks outperform graph convolutional networks on signed graphs.** Comparison of SheafNN and GCN models on the semisupervised classification task described in Section 4. Lines plotted are the mean across five random graph trials, with the standard deviation in error bars. Rows correspond to noise levels for edge weights, while columns correspond to noise levels for input features.

References

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[2] Justin Curry. *Sheaves, Cosheaves, and Applications*. PhD thesis, University of Pennsylvania, 2014.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[4] Jakob Hansen and Robert Ghrist. Learning sheaf Laplacians from smooth signals. In *Proceedings of ICASSP*, 2019.

[5] Jakob Hansen and Robert Ghrist. Toward a spectral theory of cellular sheaves. *Journal of Applied and Computational Topology*, 3(4):315–358, December 2019.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[8] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, May 2018.

[9] Amit Singer and Hau-Tieng Wu. Vector Diffusion Maps and the Connection Laplacian. *Communications in Pure and Applied Mathematics*, 65(8), 2012.

[10] S. Emre Tuna. Synchronization under matrix-weighted Laplacian. *Automatica*, 73:76–81, November 2016.

[11] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871, 2019.