Convolutional Neural Network Architectures for Signals Supported on Graphs

Fernando Gama, Antonio G. Marques, Geert Leus, and Alejandro Ribeiro

***Abstract*—Two architectures that generalize convolutional neural networks (CNNs) for the processing of signals supported on graphs are introduced. We start with the selection graph neural network (GNN), which replaces linear time invariant filters with linear shift invariant graph filters to generate convolutional features and reinterprets pooling as a possibly nonlinear subsampling stage where nearby nodes pool their information in a set of preselected sample nodes. A key component of the architecture is to remember the position of sampled nodes to permit computation of convolutional features at deeper layers. The second architecture, dubbed aggregation GNN, diffuses the signal through the graph and stores the sequence of diffused components observed by a designated node. This procedure effectively aggregates all components into a stream of information having temporal structure to which the convolution and pooling stages of regular CNNs can be applied. A multinode version of aggregation GNNs is further introduced for operation in large scale graphs. An important property of selection and aggregation GNNs is that they reduce to conventional CNNs when particularized to time signals reinterpreted as graph signals in a circulant graph. Comparative numerical analyses are performed in a source localization application over synthetic and real-world networks. Performance is also evaluated for an authorship attribution problem and text category classification. Multinode aggregation GNNs are consistently the best performing GNN architecture.**

***Index Terms*—deep learning, convolutional neural networks, graph signal processing, graph filters, pooling**

I. Introduction

We consider signals with irregular structure and describe their underlying topology with a graph whose edge weights capture a notion of expected similarity or proximity between signal components expressed at nodes [1]–[4]. Of particular importance in this paper is the interpretation of matrix representations of the graph as shift operators that can be applied to graph signals. Shift operators represent local (one-hop neighborhood) operations on the graph, and allow for different generalizations of convolution, sampling and reconstruction. These generalizations stem either from representations of graph filters as polynomials in the shift operator [1], [5], [6] or from the aggregation of sequences generated through successive application of the shift operator [7]. They not only capture the intuitive idea of convolution, sampling and reconstruction as local operations but also share some other interesting theoretical properties [1], [2], [5]. Our goal here is to build on these definitions to generalize Convolutional (C) neural networks (NNs) to graph signals.

CNNs consist of layers that are sequentially composed, each of which is itself the composition of convolution and pooling operations (Section II and Figure 1). The input to a layer is a multichannel signal composed of features extracted from the previous layer, or the input signal itself at the first layer. The main step in the convolution stage is the processing of each feature with a bank of linear time invariant filters (Section II-A). To keep complexity under control and avoid the number of intermediate features growing exponentially, the outputs of some filters are merged via simple pointwise summations. In the pooling stage we begin by computing local summaries in which feature components are replaced with a summary of their values at nearby points (Sec. II-B). These summaries can be linear, e.g., a weighted average of adjacent components, or nonlinear, e.g., the maximum value among adjacent components. Pooling also involves a subsampling of the summarized outputs. This subsampling reduces dimensionality with a (small) loss of information because the summarizing function is a low-pass operation. The output of the layer is finally obtained by application of a pointwise nonlinear activation function to produce features that become an input to the next layer. This is an architecture that is both simple to implement [8], and simple to train [9]. Most importantly, their performance in regression and classification is remarkable to the extent that CNNs have become the standard tool in machine learning to handle such inference tasks [10]–[12].

As it follows from the above description, a CNN layer involves five operations: (i) Convolution with linear time invariant filters. (ii) Summation of different features. (iii) Computation of local summaries. (iv) Subsampling. (v) Activation with a pointwise nonlinearity. A graph (G)NN is an architecture adapted to graph signals that generalizes these five operations. Operations (ii) and (v) are pointwise, therefore independent of the underlying topology, so that they can be applied without modification to graph signals. Generalizing (iii) is ready because the notion of adjacent components is well defined by graph neighborhoods. Generalization of operation (i) is not difficult in the context of graph signal processing advances whereby linear time invariant filters are particular cases of linear shift invariant graph filters. This has motivated the definition of graph (G) NNs with convolutional features computed from shift invariant graph filters, an idea that was first introduced in [13] and further explored in [14]–[19]. Architectures based on receptive fields, which are

Supported by US NSF CCF-1717120, US ARO W911NF1710438, ISTC-WAS and Intel AI DevCloud; and Spain MINECO grants No TEC2013-41604-R and TEC2016-75361-R. F. Gama and A. Ribeiro are with the Dept. of Electrical and Systems Eng., Univ. of Pennsylvania., A. G. Marques is with the Dept. of Signal Theory and Comms., King Juan Carlos Univ., G. Leus is with the Dept. of Microelectronics, Delft Univ. of Technology. Email: {fgama,aribeiro}@seas.upenn.edu, antonio.garcia.marques@urjc.es, and g.j.t.leus@tudelft.nl.

different but conceptually similar to graph filters, have also been proposed [20]–[22]. However, generalization of operation (iv) has proven more challenging because once the signal is downsampled, it is not easy to identify a coarsened graph to connect the components of the subsampled signal. The use of multiscale hierarchical clustering has been proposed to produce a collection of smaller graphs [13], [14], [16] but it is not clear which clustering or coarsening criteria is appropriate for GNN architectures. The difficulty of designing and implementing proper pooling is highlighted by the fact that several works exclude the pooling stage altogether [17], [20], [21], [23].

In this paper we propose two different GNN architectures, selection GNNs and aggregation GNNs, that include convolutional and pooling stages but bypass the need to create a coarsened graph. In selection GNNs (Sec. III and Fig. 2) we replace convolutions with linear shift invariant filters and replace regular sampling with graph selection sampling. In the first layer of the selection GNN, linear shift invariant filters are well defined as polynomials on the given graph. At the first pooling stage, however, we sample a smaller number of signal components and face the challenge of computing a graph to describe the topology of the subsampled signal. Our proposed strategy is to bypass the computation of a coarsened graph by using zero padding (Sec. III-A). This simple technique permits computation of features that are convolutional on the input graph. The pooling stage is modified to aggregate information in multihop neighborhoods as determined by the structure of the original graph and the sparsity of the subsampled signal (Sec. III-B).

In aggregation GNNs we borrow ideas from aggregation sampling [7] to create a signal with temporal structure that incorporates the topology of the graph (Sec. IV and Fig. 3). This can be accomplished by focusing on a designated node and considering the local sequence that is generated by subsequent applications of the graph shift operator. This is a signal with a temporal structure because it reflects the propagation of a diffusion process. Yet, it also captures the topology of the graph because subsequent components correspond to the aggregation of information in nested neighborhoods of increasing reach. Aggregation GNNs apply a regular CNN to the diffusion signal observed at the designated node.

We finally introduce a multinode version of aggregation GNNs, where several regular CNNs are run at several designated nodes (Sec. IV-A and Fig. 4). The resulting CNN outputs are diffused in the input graph to generate another sequence with temporal structure at a smaller subset of nodes to which regular CNNs are applied in turn. We can think of multinode aggregation GNNs as composed of inner and outer layers. Inner layers are regular CNN layers. Output layers stack CNNs joined together by a linear diffusion process. Multinode aggregation GNNs are consistently the best performing GNN architecture (Sec. V). We remark that aggregation GNNs, as well as selection GNNs are proper generalizations of conventional CNNs because they both reduce to a CNN architecture when particularized to a cyclic graph.

The proposed architectures are applied to the problems of localizing the source of a diffusion process on synthetic networks (Sec. V-A) as well as on real-world social networks (Sec. V-B). Performance is additionally evaluated on problems of authorship attribution (Sec. V-C) and classification of articles of the `20NEWS` dataset (Sec. V-D), involving real datasets. Results are compared to those obtained from a graph coarsening architecture using a multiscale hierarchical clustering scheme [16]. The results are encouraging and show that the multinode approach consistently outperforms the other architectures.

Notation: The n-th component of a vector $\mathbf{x}$ is denoted as $[\mathbf{x}]_n$. The (m,n) entry of a matrix $\mathbf{X}$ is $[\mathbf{X}]_{mn}$. The vector $\mathbf{x} := [\mathbf{x}_1; \ldots; \mathbf{x}_n]$ is a column vector stacking the column vectors $\mathbf{x}_n$. When $\mathbf{n}$ denotes a set of subindices, $|\mathbf{n}|$ is the number of elements in $\mathbf{n}$ and $[\mathbf{x}]_{\mathbf{n}}$ denotes the column vector formed by the elements of $\mathbf{x}$ whose subindices are in $\mathbf{n}$. The vector $\mathbf{1}$ is the all-ones vector.

II. Convolutional Neural Networks

Given a training set $\mathcal{T} := \{(\mathbf{x}, \mathbf{y})\}$ formed by inputs $\mathbf{x}$ and their associated outputs $\mathbf{y}$, a learning algorithm produces a representation (mapping) that can estimate the output $\hat{\mathbf{y}}$ that should be assigned to an input $\hat{\mathbf{x}} \notin \mathcal{T}$. NNs produce a representation using a stacked layered architecture in which each layer composes a linear transformation with a pointwise nonlinearity [24]. Formally, the first layer of the architecture begins with a linear transformation to produce the intermediate output $\mathbf{u}_1 := \mathbf{A}_1\mathbf{x}_0 = \mathbf{A}_1\hat{\mathbf{x}}$ followed by a pointwise nonlinearity to produce the first layer output $\mathbf{x}_1 := \sigma_1(\mathbf{u}_1) = \sigma_1(\mathbf{A}_1\mathbf{x}_0)$. This procedure is applied recursively so that at the ℓth layer we compute the transformation

$$\mathbf{x}_\ell := \sigma_\ell(\mathbf{u}_\ell) := \sigma_\ell(\mathbf{A}_\ell\mathbf{x}_{\ell-1}). \tag{1}$$

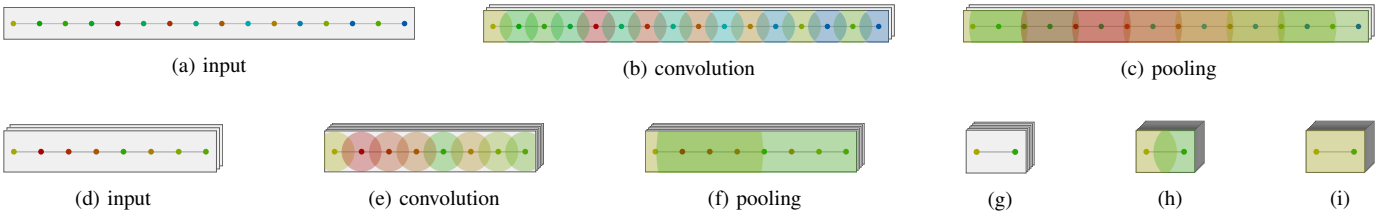
In an architecture with L layers, the input $\hat{\mathbf{x}} = \mathbf{x}_0$ is fed to the first layer and the output $\hat{\mathbf{y}} = \mathbf{x}_L$ is read from the last layer [25]. Elements of the training set $\mathcal{T}$ are used to find matrices $\mathbf{A}_\ell$ that optimize a training cost of the form $\sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} f(\mathbf{y}, \mathbf{x}_L)$, where $f(\mathbf{y}, \mathbf{x}_L)$ is a fitting metric that assess the difference between the NN's output $\mathbf{x}_L$ produced by input $\mathbf{x}$ and the desired output $\mathbf{y}$ stored in the training set. Computation of the optimal NN coefficients $\mathbf{A}_\ell$ is typically carried out by stochastic gradient descent, which can be efficiently computed using the backpropagation algorithm [9].

The NN architecture in (1) is a multilayer perceptron composed of fully connected layers [25]. If we denote as M_ℓ the number of entries of the output of layer ℓ, the matrix $\mathbf{A}_\ell$ contains $M_\ell \times M_{\ell-1}$ components. This, likely extremely, large number of parameters not only makes training challenging but empirical evidence suggests that it leads to overfitting [26]. CNNs resolve this problem with the introduction of two operations: Convolution and pooling.

A. Convolutional Features

To describe the creation of convolutional features write the output of the $(\ell-1)$st layer as $\mathbf{x}_{\ell-1} := [\mathbf{x}^1_{\ell-1}; \ldots; \mathbf{x}^{F_{\ell-1}}_{\ell-1}]$. This decomposes the $M_{\ell-1}$-dimensional output of the $(\ell-1)$st layer as a stacking of $F_{\ell-1}$ features of dimension $N_{\ell-1}$. This collection of features is the input to the ℓth layer. Likewise,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSP.2018.2887403, IEEE Transactions on Signal Processing



(a) input (b) convolution (c) pooling

(d) input (e) convolution (f) pooling (g) (h) (i)

Figure 1. Convolutional Neural Networks. (a) Consider the input to be a discrete time signal, represented by a succession of signal values. (b) Convolve this signal with a filter to obtain corresponding features [cf. (2)]. The color disks centered at each node symbolize the convolution operation. (c) Apply pooling [cf. (4)]. The color disks symbolize the reach of the pooling operation (the number of samples that are pooled together) (d) Downsample to obtain a discrete time signal of smaller size [cf. (5)]. (e)-(i) Repeat the application of convolution and pooling, trading off the temporal dimension for more features.

the intermediate output $\mathbf{u}_\ell$ can be written as a collection of F_ℓ features $\mathbf{u}_\ell := [\mathbf{u}_\ell^1; \ldots; \mathbf{u}_\ell^{F_\ell}]$ where $\mathbf{u}_\ell^f$ is of length $N_{\ell-1}$ and is obtained through convolution and linear aggregation of features $\mathbf{x}_{\ell-1}^g$ of the previous layer, $g = 1, \ldots, F_{\ell-1}$. Specifically, let $\mathbf{h}_\ell^{fg} := [\, [\mathbf{h}_\ell^{fg}]_0; \ldots; [\mathbf{h}_\ell^{fg}]_{K_\ell-1} \,]$ be the coefficients of a K_ℓ-tap linear time invariant filter that is used to process the gth feature of the $(\ell-1)$st layer to produce the intermediate feature $\mathbf{u}_\ell^{fg}$ at layer ℓ. Since the filter is defined by a convolution, the components of $\mathbf{u}_\ell^{fg}$ are explicitly given by

$$\left[\mathbf{u}_\ell^{fg}\right]_n := \left[\mathbf{h}_\ell^{fg} * \mathbf{x}_{\ell-1}^g\right]_n = \sum_{k=0}^{K_\ell-1} \left[\mathbf{h}_\ell^{fg}\right]_k \left[\mathbf{x}_{\ell-1}^g\right]_{n-k}, \quad (2)$$

where we consider that: i) the output has the same size than the input and ii) the convolution (2) is circular to account for border effects. After evaluating the convolutions in (2), the ℓth layer features $\mathbf{u}_\ell^f$ are computed by aggregating the intermediate features $\mathbf{u}_\ell^{fg}$ associated with each of the previous layer features $\mathbf{x}_{\ell-1}^g$ using a simple summation,

$$\mathbf{u}_\ell^f := \sum_{g=1}^{F_{\ell-1}} \mathbf{u}_\ell^{fg} = \sum_{g=1}^{F_{\ell-1}} \mathbf{h}_\ell^{fg} * \mathbf{x}_{\ell-1}^g. \quad (3)$$

The vector $\mathbf{u}_\ell := [\mathbf{u}_\ell^1; \ldots; \mathbf{u}_\ell^{F_\ell}]$ obtained from (2) and (3) represents the output of the linear operation of the ℓth layer of the CNN [cf. (1)]. Although not explicitly required, the number of features F_ℓ and the number of filter taps K_ℓ are typically much smaller than the dimensionality $M_{\ell-1}$ of the features $\mathbf{x}_{\ell-1}$ that are processed by the ℓth layer. This reduces the number of learnable parameters from $M_\ell \times M_{\ell-1}$ in (1) to $K_\ell \times F_\ell \times F_{\ell-1}$ simplifying training and reducing overfitting.

B. Pooling

The features $\mathbf{u}_\ell^{fg}$ in (2) and their consolidated counterparts $\mathbf{u}_\ell^f$ in (3) have $N_{\ell-1}$ components. This number of components is reduced to N_ℓ at the pooling stage in which the values of a group of neighboring elements are aggregated to a single scalar using a possibly nonlinear summarization function ρ_ℓ. To codify the locality of ρ_ℓ, we define, with a slight abuse of notation, $\mathbf{n}_\ell$ as a vector containing the indexes associated with index n – e.g., use $\mathbf{n}_\ell = [n-1; n; n+1]$ to group adjacent components – and define the signal $\mathbf{v}_\ell^f$ with components

$$\left[\mathbf{v}_\ell^f\right]_n = \rho_\ell\left(\left[\mathbf{u}_\ell^f\right]_{\mathbf{n}_\ell}\right). \quad (4)$$

The summarization function ρ_ℓ in (4) acts as a low-pass operation and the most common choices are the maximum $\rho_\ell([\mathbf{u}_\ell^f]_{\mathbf{n}_\ell}) = \max([\mathbf{u}_\ell^f]_{\mathbf{n}_\ell})$ and the average $\rho_\ell([\mathbf{u}_\ell^f]_{\mathbf{n}_\ell}) = \mathbf{1}^\mathsf{T}[\mathbf{u}_\ell^f]_{\mathbf{n}_\ell}/|\mathbf{n}_\ell|$ [27].

To complete the pooling stage we follow (4) with a downsampling operation. For that matter, we define the sampling matrix $\mathbf{C}_\ell$ as a fat binary matrix with $N_{\ell-1}$ columns and N_ℓ rows, which are selected from the rows of the identity matrix. When the sampling matrix $\mathbf{C}_\ell$ is *regular*, the nonzero entries follow the pattern $[\mathbf{C}_\ell]_{mn} = 1$ if n can be written as $n = (N_{\ell-1}/N_\ell)m$ and zero otherwise; hence, the product $\mathbf{C}_\ell \mathbf{v}_\ell^f$ selects one out of every $(N_{\ell-1}/N_\ell)$ components of $\mathbf{v}_\ell^f$. Downsampling is composed with a pointwise nonlinearity to produce the ℓth layer features

$$\mathbf{x}_\ell^f = \sigma_\ell\left(\mathbf{C}_\ell \mathbf{v}_\ell^f\right). \quad (5)$$

The compression or downsampling factor $(N_{\ell-1}/N_\ell)$ is often matched to the local summarization function ρ_ℓ so that the set $\mathbf{n}_\ell$ contains $(N_{\ell-1}/N_\ell)$ adjacent indexes. We further note that although we defined (4) for all n, in practice, we only compute the components of $\mathbf{v}_\ell^f$ that are to be selected by the sampling matrix $\mathbf{C}_\ell$. In fact, it is customary to combine (4) and (5) to simply write $[\mathbf{x}_\ell^f]_n = \sigma_l(\rho_\ell([\mathbf{u}_\ell^f]_{\mathbf{n}_\ell})$ for n in the selection set. Separating the nonlinearity in (4) from the downsampling operation in (5) is convenient to elucidate pooling strategies for signals on graphs.

Equations (2)-(5) complete the specification of the CNN architecture. We begin at each layer with the input $\mathbf{x}_{\ell-1} := [\mathbf{x}_{\ell-1}^1; \ldots; \mathbf{x}_{\ell-1}^{F_{\ell-1}}]$. Features are fed to parallel convolutional channels to produce the features $\mathbf{u}_\ell^{fg}$ in (2) and consolidated into the features $\mathbf{u}_\ell^f$ in (3). These features are fed to the local summarization function ρ_ℓ to produce features $\mathbf{v}_\ell^f$ [cf. (4)] which are then downsampled and processed by the pointwise activation nonlinearity σ_ℓ to produce the features $\mathbf{x}_\ell^f$ [cf. (5)]. The output of the ℓth layer is the vector $\mathbf{x}_\ell := [\mathbf{x}_\ell^1; \ldots; \mathbf{x}_\ell^{F_\ell}]$ that groups the features in (5). We point out for completeness that the Lth layer is often a fully connected layer in the mold of (1) that does not abide to the convolutional and pooling paradigm of (2)-(5). Thus, the Lth layer produces an arbitrary (non convolutional) linear combination of F_{L-1} features to produce the final F_L scalar features $\mathbf{x}_L$. The output of this readout layer provides the estimate $\hat{\mathbf{y}} = \mathbf{x}_L$ that is associated with the input $\hat{\mathbf{x}} = \mathbf{x}_0$ fed to the first layer.

C. Signals on Graphs

There is overwhelming empirical evidence that CNNs are superb representations of signals defined in regular domains such as time series and images [10]. Our goal in this paper is to contribute to the extension of these architectures to signals supported in irregular domains described by arbitrary graphs. Consider then a weighted graph with N nodes, edge set $\mathcal{E}$ and weight function $\mathcal{W} : \mathcal{E} \to \mathbb{R}$. We endow the graph with a shift operator $\mathbf{S}$, which is an $N \times N$ square matrix having the same sparsity pattern of the graph; i.e., we can have $[\mathbf{S}]_{mn} \neq 0$ if and only if $(n, m) \in \mathcal{E}$ or $m = n$. The shift operator is a stand in for one of the matrix representations of the graph. Commonly used shift operators include the adjacency matrix $\mathbf{A}$ with nonzero elements $[\mathbf{A}]_{mn} = \mathcal{W}(n, m)$ for all $(n, m) \in \mathcal{E}$, the Laplacian $\mathbf{L} := \text{diag}(\mathbf{A1}) - \mathbf{A}$ and their normalized counterparts $\bar{\mathbf{A}}$ and $\bar{\mathbf{L}}$ [3].

Consider the signal $\mathbf{x} = [\mathbf{x}^1; \ldots; \mathbf{x}^F]$ formed by F feature vectors $\mathbf{x}^f$ with N components each. The feature vector $\mathbf{x}^f$ is said to be a graph signal when each of its N components is assigned to a different vertex of the graph. The graph describes the underlying support of the data $\mathbf{x}^f$ (hence, of $\mathbf{x}$) by using the weights $\mathcal{W}$ to encode arbitrary pairwise relationships between data elements. The graph shift enables processing of the graph signal $\mathbf{x}^f$ because it defines a local linear operation that can be applied to graph signals. Indeed, if we consider the signal $\mathbf{y}^f := \mathbf{S}\mathbf{x}^f$ it follows from the sparsity of $\mathbf{S}$ that the nth element of $\mathbf{y}^f$ depends on the elements of $\mathbf{x}^f$ associated with neighbors of the node n,

$$[\mathbf{y}^f]_n = \sum_{m:(m,n)\in\mathcal{E}} [\mathbf{S}]_{nm}[\mathbf{x}^f]_m. \tag{6}$$

It is instructive to consider the cyclic graph adjacency matrix $\mathbf{A}_{\text{dc}}$, with nonzero elements $[\mathbf{A}_{\text{dc}}]_{1+n \mod N,n} = 1$. Since the cyclic graph describes the structure of discrete (periodic) time, we can say that a discrete time signal $\mathbf{x}$ is a graph signal defined on the cyclic graph. When particularized to $\mathbf{S} = \mathbf{A}_{\text{dc}}$, (6) yields $y^f_{1+n \mod N} = x^f_n$ implying that $\mathbf{y}^f$ is a circularly time shifted copy of $\mathbf{x}^f$. This motivates interpretation of $\mathbf{S}$ as the generalization of time shifts to signals supported in the corresponding graph [1].

Enabling CNNs to process data modeled as graph signals entails extending the operations of convolution and pooling to handle the irregular nature of the underlying support. Convolution [cf. (2)] can be readily replaced by the use of linear, shift invariant graph filters [cf. (7)]. The summarizing function [cf. (4)] can also be readily extended by using the notion of neighborhood defined by the underlying graph support. The pointwise nonlinearity can be kept unmodified [cf. (5)], but there are two general downsampling strategies for graph signals: selection sampling [28] and aggregation sampling [7]. Inspired by these, we propose two architectures: selection GNNs (Section III) and aggregation (Section IV) GNNs.

Remark 1. Although our current theoretical understanding of CNNs is limited, empirical evidence suggests that convolution and pooling work in tandem to act as feature extractors at different levels of resolution. At each layer, the convolution operation linearly relates up to K_ℓ nearby values of each input feature. Since the same filter taps are used to process the whole signal, the convolution finds patterns that, albeit local, are irrespective of the specific location of the pattern in the signal. The use of several features allows collection of different patterns through learning of different filters thus yielding a more expressive operation. The pooling stage summarizes information into a feature of lower dimensionality. It follows that subsequent convolutions operate on summaries of different regions. As we move into deeper layers we pool summaries of summaries that are progressively growing the region of the signal that affects a certain feature. The conjectured value of composing local convolutions with pooling summaries is adopted prima facie as we seek graph neural architectures that exploit the locality of the shift operator to generalize convolution and pooling operations.

III. Selection Graph Neural Networks

Generalizing the first layer of a CNN to signals supported on graphs is straightforward as it follows directly from the definition of a linear shift invariant filter [5]. Going back to the definition of convolutional features in (2) we reinterpret the filters $\mathbf{h}_1^{fg}$ as graph filters that process the features $\mathbf{x}_0^g$ through a graph convolution. This results in intermediate features $\mathbf{u}_1^{fg}$ having components

$$\left[\mathbf{u}_1^{fg}\right]_n := \left[\mathbf{h}_1^{fg} *_{\mathbf{S}} \mathbf{x}_0^g\right]_n := \sum_{k=0}^{K_1-1} \left[\mathbf{h}_1^{fg}\right]_k \left[\mathbf{S}^k\mathbf{x}_0^f\right]_n, \tag{7}$$

where we have used $*_{\mathbf{S}}$ to denote the graph convolution operation on $\mathbf{S}$. The summations in equations (2) and (7) are analogous except for the different interpretations of what it means to shift the input signal $\mathbf{x}_0^f$. In (2), a k-unit shift at index n means considering $[\mathbf{x}_0^f]_{n-k}$, the value of the signal $\mathbf{x}_0^f$ at time $n-k$. In (7), graph shifting at node n entails the operation $[\mathbf{S}^k\mathbf{x}_0^f]_n$ which composes a multiplication by $\mathbf{S}^k$ with the selection of the resulting value at n. In fact, particularizing (7) to the cyclic graph by making $\mathbf{S} = \mathbf{A}_{\text{dc}}$ renders (2) and (7) equivalent. From the perspective of utilizing (7) as an extractor of local (graph) convolutional features it is important to note that graph convolutions aggregate information through successive local operations [cf. (6)]. A filter with K_1 taps incorporates information at node n that comes from nodes in its $(K_1 - 1)$-hop neighborhood.

Although we wrote (7) componentwise to emphasize its similarity with (2) we can drop the n subindices to write a vector relationship. For future reference we further define the linear shift invariant filter $\mathbf{H}_1^{fg} := \sum_{k=0}^{K_1-1} [\mathbf{h}_1^{fg}]_k \mathbf{S}^k$ to write

$$\mathbf{u}_1^{fg} = \sum_{k=0}^{K_1-1} \left[\mathbf{h}_1^{fg}\right]_k \mathbf{S}^k\mathbf{x}_0^f := \mathbf{H}_1^{fg}\mathbf{x}_0^f. \tag{8}$$

The graph filter (8) is a generalization of the Chebyshev filter in [16]. More precisely, if $\mathcal{G}$ is an undirected graph, and we adopt the normalized Laplacian as the graph shift operator $\mathbf{S}$, then (8) boils down to a Chebyshev filter. The convolutional stage in [18] is a Chebyshev filter of $K = 2$, and thus is also a special case of (8). We also note that the use of polynomials on arbitrary graph shift operators for the convolutional stage

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSP.2018.2887403, IEEE Transactions on Signal Processing

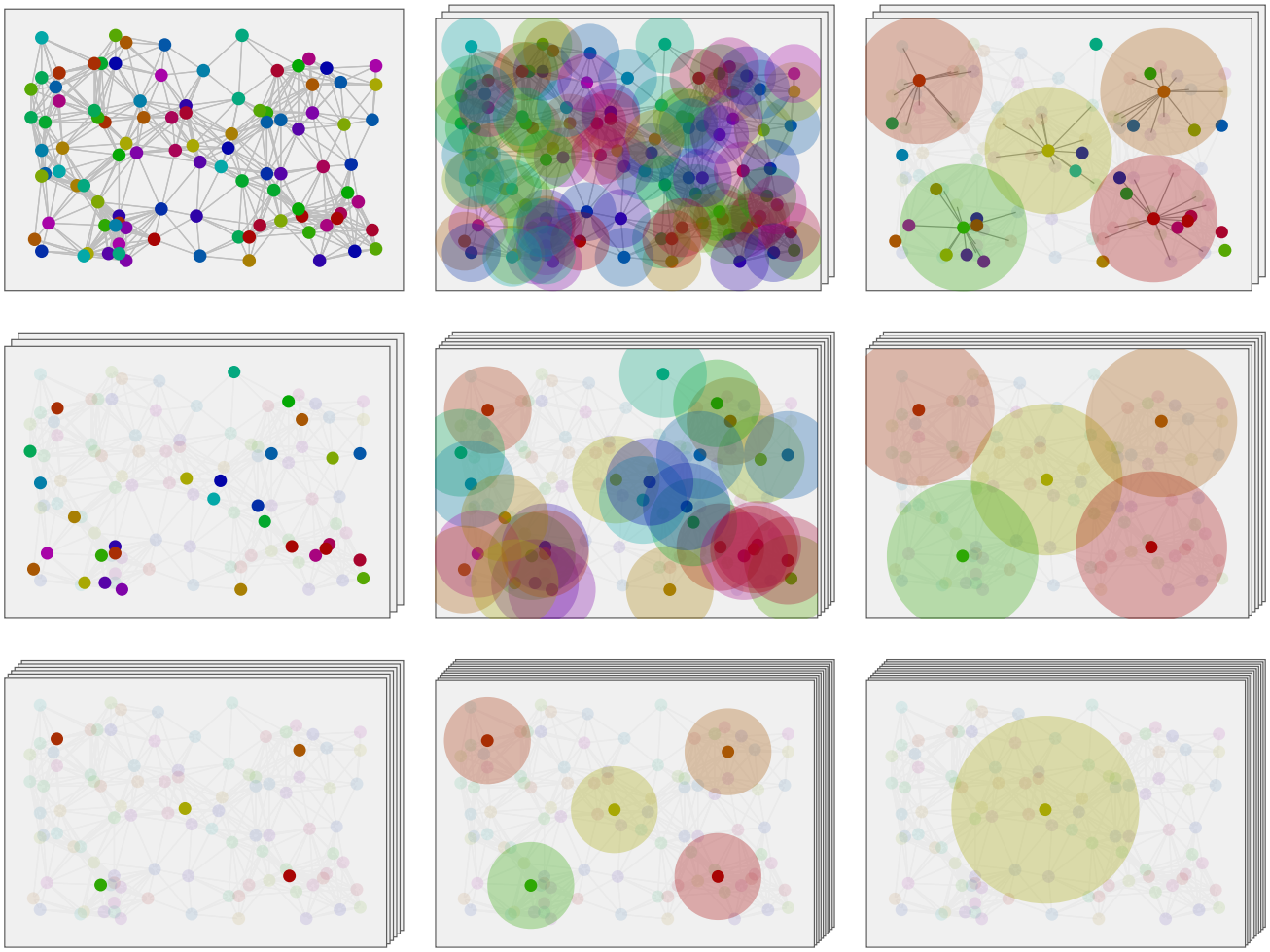


Figure 2. Selection Graph Neural Networks. Consider the input to be a signal supported by a known N-node graph. First, convolutional features are obtained by means of graph filtering in the original graph [cf. (8)]. The color disks in the second column illustrate the convolution operation on each node. Then, a subset of N_1 nodes is selected, and summarizing function ρ_1 and pointwise nonlinearity σ_1 are applied to the neighborhood $\mathbf{n}_1$ for each of these nodes, obtaining the output $\mathbf{x}_1^f$ for the first layer. The color disks in the third column show the reach of the pooling operation, the size of the neighborhood being pooled (in the first row, the disks include only the one-hop neighborhood; also, only a few disks are shown so as not to clutter the illustration). In order to obtain convolutional features for following layers, we zero pad the signal to *fit* the original graph [cf. (9)] so as to apply a graph filter and then resample the output at the same set of nodes [cf. (11)-(13)]. Then, a new smaller subset of nodes is selected, and the summarizing function and pointwise nonlinearity are applied to a neighborhood of these nodes [cf. (15)]. This process is repeated while selecting fewer and fewer nodes.

has been also proposed in [17], [23]. Asides from replacing the linear time invariant filter in (2) with the graph shift invariant filter in (8), the remaining components of the conventional CNN architecture can remain more or less unchanged. The feature aggregation in (3) to obtain $\mathbf{u}_1^f$ needs no modification as it is a simple summation independent of the graph structure. The summarization operator in (4) requires a redefinition of locality. This is not difficult because it follows from (8) that $\mathbf{u}_1^f$ is another N-node graph signal that is defined over the same graph $\mathbf{S}$. We can then use $\mathbf{n}_1$ to represent a graph neighborhood of node n and apply the same summary operator. We point out that $\mathbf{n}_1$ need not be the 1-hop neighborhood of n. The sampling and activation operation in (5) requires a matrix $\mathbf{C}_1$ to sample over the irregular graph domain. Apart from the challenge of selecting sampling matrices for graphs – see (16) and [7], [28]–[30] –, this does not require any further modification to (5). The first row of Fig. 2 shows the operations carried out in this first layer.

The challenge in generalizing CNNs to GNNs arises beyond the first layer. After implementing the sampling operation in (5) the signal $\mathbf{x}_1^f$ is of lower dimensionality than $\mathbf{u}_1^f$ and can no longer be interpreted as a signal supported on $\mathbf{S}$. In regular domains this is not a problem because we use the extraneous geometrical information of the underlying domain to define convolutions in the space of lower dimensionality. To see this in terms of graph signals, let us interpret the signal $\mathbf{x}_0^g$ defined on a regular domain as one defined on a cyclic graph with $N_0 = N$ nodes, which is also the same graph that describes $\mathbf{u}_1^f$. Then, if we consider a downsampling factor of (N_1/N_0), another *cyclic* graph with N_1 nodes describes the signal $\mathbf{x}_1^f$. However, when graph signals are defined in a generic irregular domain, there is no extraneous information to

elucidate the form of the graph that describes signals beyond the first layer. Resolving mismatched supports is a well-known problem in signal processing whose simplest and most widely-used solution is zero padding. The following sections illustrate how zero padding can be leveraged to resolve one of the critical challenges in the implementation of GNNs.

A. Selection Sampling on Graph Convolutional Features

Sampling is an operation that selects components of a signal. To explain the construction of convolutional features on graphs, it is more convenient to think of sampling as the *selection of nodes* of a graph which we call active nodes. This implies that at each layer ℓ we place the input features $\mathbf{x}_{\ell-1}^{f}$ of dimension $N_{\ell-1}$ on top of the active nodes of the graph $\mathbf{S}$. Selection schemes are further discussed in Sec. III-C. Doing so requires that we keep track of the location of the samples. Thus, at each layer ℓ we consider input features $\mathbf{x}_{\ell-1}^{g}$ each with $N_{\ell-1}$ components, and zero padded features $\tilde{\mathbf{x}}_{\ell-1}^{g}$ each with size N but only $N_{\ell-1}$ nonzero components which replicate the values of $\mathbf{x}_{\ell-1}^{g}$. The indexes of the nonzero components of $\tilde{\mathbf{x}}_{\ell-1}^{g}$ correspond to the location of the elements of $\mathbf{x}_{\ell-1}^{g}$ in the original graph. It is clear that we can move from the unpadded to the padded representation by multiplying with an $N \times N_{\ell-1}$ tall binary sampling matrix $\mathbf{D}_{\ell-1}^{\mathsf{T}}$. Indeed, if we let $[\mathbf{D}_{\ell-1}]_{mn} = 1$ represent the mth component of the unpadded feature, $[\mathbf{x}_{\ell-1}^{g}]_{m}$, is located in the nth node of the graph, we can write the padded feature as

$$\tilde{\mathbf{x}}_{\ell-1}^{g} \;=\; \mathbf{D}_{\ell-1}^{\mathsf{T}} \mathbf{x}_{\ell-1}^{g}. \tag{9}$$

The advantage of keeping track of the padded signal is that convolutional features can be readily obtained by operating in the original graph. Given the notion of graph convolution in (8) and (re-)defining $\mathbf{h}_{\ell}^{fg}$ to be the graph filter coefficients at layer ℓ we can define intermediate features as [cf. (2)]

$$\tilde{\mathbf{u}}_{\ell}^{fg} \;:=\; \sum_{k=0}^{K_{\ell}-1} \left[\mathbf{h}_{\ell}^{fg}\right]_{k} \mathbf{S}^{k}\, \tilde{\mathbf{x}}_{\ell-1}^{g}. \tag{10}$$

Although a technical solution to the construction of convolutional features, (10) does not exploit the computational advantages of sampling. These can be recovered by selecting components of $\tilde{\mathbf{u}}_{\ell}^{fg}$ at the same set of nodes that support $\mathbf{x}_{\ell-1}^{g}$. We then define $\mathbf{u}_{\ell}^{fg} := \mathbf{D}_{\ell-1}\tilde{\mathbf{u}}_{\ell}^{fg}$. If we further use (9) to substitute $\tilde{\mathbf{x}}_{\ell-1}^{g}$ into the definition of the convolutional features in (10), we can write

$$\mathbf{u}_{\ell}^{fg} := \mathbf{D}_{\ell-1}\tilde{\mathbf{u}}_{\ell}^{fg} = \mathbf{D}_{\ell-1} \sum_{k=0}^{K_{\ell}-1} \left[\mathbf{h}_{\ell}^{fg}\right]_{k} \mathbf{S}^{k}\, \mathbf{D}_{\ell-1}^{\mathsf{T}}\, \mathbf{x}_{\ell-1}^{g}. \tag{11}$$

If we further define reduced dimensionality k-shift matrices

$$\mathbf{S}_{\ell}^{(k)} := \mathbf{D}_{\ell-1}\, \mathbf{S}^{k}\, \mathbf{D}_{\ell-1}^{\mathsf{T}}, \tag{12}$$

and reorder and regroup terms in (11) we can reduce the definition of convolutional features to

$$\mathbf{u}_{\ell}^{fg} \;=\; \sum_{k=0}^{K_{\ell}-1} \left[\mathbf{h}_{\ell}^{fg}\right]_{k} \mathbf{S}_{\ell}^{(k)}\, \mathbf{x}_{\ell-1}^{g} \;=\; \mathbf{H}_{\ell}^{fg} \mathbf{x}_{\ell-1}^{g}, \tag{13}$$

where we have also defined the *subsampled* linear shift invariant filter $\mathbf{H}_{\ell}^{fg} := \sum_{k=0}^{K_{\ell}-1} [\mathbf{h}_{\ell}^{fg}]_{k} \mathbf{S}_{\ell}^{(k)}$. Implementing (11) entails repeated application of the shift operator to the padded signal, which can be carried out with low cost if the original input graph is sparse. In (13), the filter $\mathbf{H}_{\ell}^{fg}$ takes advantage of sampling to operate directly on a space of lower dimension $N_{\ell-1}$. The matrices $\mathbf{S}_{\ell}^{(k)}$ can be computed beforehand because they depend on the graph shift operator and the sampling matrices only. We emphasize that, save for subsampling, (13) and (11) are equivalent and that, therefore, the features $\mathbf{u}_{\ell}^{fg}$ generated by the subsampled filter $\mathbf{H}_{\ell}^{fg}$ are convolutional relative to the original graph (shift) $\mathbf{S}$. The middle image in Fig. 2 shows zero pad of input signal, convolution in the original graph, and resampling of the filter output.

Features $\mathbf{u}_{\ell}^{f}$ can be obtained from features $\mathbf{u}_{\ell}^{fg}$ using the same linear aggregation operation in (3) which does not require adaptation to the structure of the graph,

$$\mathbf{u}_{\ell}^{f} \;=\; \sum_{g=1}^{F_{\ell-1}} \mathbf{H}_{\ell}^{fg} \mathbf{x}_{\ell-1}^{g}. \tag{14}$$

This completes the construction of convolutional features and leads to the pooling stage we describe next.

B. Selection Sampling and Pooling

The pooling stage requires that we redefine the summary and sampling operations in (4) and (5). Generalizing the summary operation requires redefining the aggregation neighborhood. In the first layer, this can be readily accomplished by selecting the α_1-hop neighborhood of each node for some given α_1 that defines the reach of the summary operation. This information is actually contained in the powers of the shift operator. The 1-hop neighborhood of n is the set of nodes m such that $[\mathbf{S}]_{nm} \neq 0$, the 2-hop neighborhood is the union of this set with those nodes m with $[\mathbf{S}^2]_{nm} \neq 0$ and so on. In the case of the sampled features the graph neighborhoods need to be intersected with the set of active nodes. This intersection is already captured by the k-shift matrices $\mathbf{S}_{\ell}^{(k)}$ [cf. (12)]. Thus, at layer ℓ we introduce an integer α_{ℓ} to specify the reach of the summary operator and define the α_{ℓ}-hop neighborhood of n as

$$\mathbf{n}_{\ell} = \left[m : \left[\mathbf{S}_{\ell}^{(k)}\right]_{nm} \neq 0, \text{ for some } k \leq \alpha_{\ell}\right]. \tag{15}$$

Summary features $[\mathbf{v}_{\ell}^{f}]_{n}$ at node n are computed from (4) using the graph neighborhoods in (15). These neighborhoods follow the node proximity encoded by $\mathbf{S}$, see third column of Fig. 2.

To formally explain the downsampling operation in (5) in the context of graph signals, begin by defining sampling matrices adapted to irregular domains. This can be easily defined at the ℓth layer if we let the sampling matrix $\mathbf{C}_{\ell}$ be a fat matrix with N_{ℓ} rows and $N_{\ell-1}$ columns with the properties

$$[\mathbf{C}_{\ell}]_{mn} \in \{0, 1\}, \quad \mathbf{C}_{\ell}\mathbf{1} = \mathbf{1}, \quad \mathbf{C}_{\ell}^{\mathsf{T}}\mathbf{1} \leq \mathbf{1}. \tag{16}$$

When $[\mathbf{C}_{\ell}]_{mn} = 1$ it means that the nth component of $\mathbf{v}_{\ell}^{f}$ is selected in the product $\mathbf{C}_{\ell}\mathbf{v}_{\ell}^{f}$ and stored as the mth component

of the output. The properties in (16) ensure that exactly N_ℓ components of $\mathbf{v}_\ell^f$ are selected and that no component is selected more than once. They do not, however, enforce a regular sampling pattern. We further define the nested sampling matrix $\mathbf{D}_\ell$ as the product of all sampling matrices applied up until layer ℓ

$$\mathbf{D}_\ell \;=\; \mathbf{C}_\ell \mathbf{C}_{\ell-1} \dots \mathbf{C}_1 \;=\; \prod_{\ell'=1}^{\ell} \mathbf{C}_{\ell'}. \tag{17}$$

Matrix $\mathbf{D}_\ell$ keeps track of the location of the selected nodes in the original graph, for each layer ℓ, and is thus used for the zero padding operation in (11).

Each layer of the selection GNN architecture is determined by (13)-(14) for the convolution operation and (4)-(5) for pooling and nonlinearity. To summarize, the input to layer ℓ is $\mathbf{x}_{\ell-1}$ comprised of $F_{\ell-1}$ features $\mathbf{x}_{\ell-1}^f$ located at a subset of nodes given by $\mathbf{D}_{\ell-1}$. Then, we use the reduced dimensionality k-shift matrices (12) to process $\mathbf{x}_{\ell-1}^f$ using a graph filter as in (13), and obtain aggregated features $\mathbf{u}_\ell^f$ (14). A neighborhood $\mathbf{n}_\ell$ for each element of $\mathbf{u}_f$ is determined by (15) for some α_ℓ and the output $\mathbf{v}_\ell^f$ of the summarizing function ρ_ℓ is computed as in (4). Finally, following (5), a smaller subset of nodes is selected by means of $\mathbf{C}_\ell$ and the pointwise nonlinearity σ_ℓ is applied to obtain the ℓth output features $\mathbf{x}_\ell^f$, for $f = 1, \dots, F_\ell$. See Algorithm 1 for details.

Remark 2. The selection GNN architecture recovers a conventional CNN when particularized to graph signals described by a cyclic graph (conventional discrete time signals). To see this, let $\mathbf{S} = \mathbf{A}_{\text{dc}}$ for a graph of size N, and let $\mathbf{C}_{\ell-1}$ be the sampling matrix that takes $N_{\ell-1}$ equally spaced samples out of the previous $N_{\ell-2}$ samples, for every ℓ. Then, the nested sampling matrix $\mathbf{D}_{\ell-1}$ becomes a sampling matrix that takes $N_{\ell-1}$ equally spaced samples out of the N original ones. This implies that $\mathbf{S}_\ell^{(k)} = \mathbf{D}_{\ell-1}\mathbf{A}_{\text{dc}}^k\mathbf{D}_{\ell-1}^\mathsf{T}$ becomes either the kth power of the adjacency matrix of a cyclic graph with $N_{\ell-1}$ nodes for k a multiple of $N/N_{\ell-1}$, or the all-zero matrix otherwise. This results in convolutional features obtained by (13) being equivalent to those obtained by (2). Likewise, making $\alpha_\ell = N_{\ell-1}/N_\ell$ for all ℓ leads to regular pooling and downsampling. This shows that the selection GNN does indeed boil down to the conventional CNN for discrete time signals.

Remark 3. The dimension N_ℓ is being effectively reduced without the need to use a complex multiscale hierarchical clustering algorithm. More specifically, in each layer, only a new set of nodes is used, but there is no need to recompute edges between these nodes or new weight functions, since the underlying graph on which the operations are actually carried out is the same graph support as the initial input data $\mathbf{x}$. This, not only avoids the computational cost of obtaining multiscale hierarchical clusters, but also avoids the need to assess when such clustering scheme is adequate.

C. Practical Considerations

Algorithm 1 Selection Graph Neural Network.

Input: $\{\hat{\mathbf{x}}\}$: testing dataset, $\mathcal{T}$: training dataset
$\mathbf{S}$: graph shift operator, L: Number of layers,
$\{F_\ell\}$: number of features, $\{K_\ell\}$: degree of filters
$\{\rho_\ell\}$: neighborhood summarizing function
`selection`: selection sampling method
$\{N_\ell\}$: number of nodes on each layer
$\{\sigma_\ell\}$: pointwise nonlinearity
Output: $\{\hat{\mathbf{y}}\}$: estimates of $\{\hat{\mathbf{x}}\}$

procedure SELECTION_GNN($\{\hat{\mathbf{x}}\}$, $\mathcal{T}$, $\mathbf{S}$, L, $\{F_\ell\}$,$\{K_\ell\}$, $\{\rho_\ell\}$, `selection`, $\{N_\ell\}$, $\{\sigma_\ell\}$)
▷ *Create architecture:*
  for $\ell = 1 : L-1$ **do**
    Compute $\mathbf{D}_{\ell-1} = \mathbf{C}_{\ell-1}\mathbf{D}_{\ell-2}$ ▷ *See* (17)
    Compute $\mathbf{S}_\ell^{(k)}$ for $k = 0, \dots, K_\ell - 1$ ▷ *See* (12)
    Create $[\mathbf{h}_\ell^{fg}]_k$, $f = 1, \dots, F_\ell$, $g = 1, \dots, F_{\ell-1}$
    Compute filters $\mathbf{H}_\ell^{fg} = \sum_{k=0}^{K_{\ell-1}} [\mathbf{h}_\ell^{fg}]_k \mathbf{S}_\ell^{(k)}$
    Aggregate filtered features $\sum_{g=1}^{F_{\ell-1}} (\mathbf{H}_\ell^{fg} \cdot)$
    Apply summarizing function $\rho_\ell(\cdot)$
    Select N_ℓ nodes following method `selection`

$$\mathbf{C}_\ell = \texttt{selection}(N_\ell, \mathbf{C}_{\ell-1})$$

    Downsample output of summarizing function $\mathbf{C}_\ell \rho_\ell$
    Apply pointwise nonlinearity $\sigma_\ell(\cdot)$
  end for
  Create fully connected layer $\mathbf{A}_L \cdot$
▷ *Train:*
  Learn $\{[\mathbf{h}_\ell^{fg}]_k\}$ and $\mathbf{A}_L$ from $\mathcal{T}$
▷ *Evaluate:*
  Obtain $\hat{\mathbf{y}}$ applying GNN on $\hat{\mathbf{x}}$ with learned coefficients
end procedure

Selection of nodes. There is a vast GSP literature on sampling by selecting nodes, see, e.g., [28]–[32]. In this paper, we consider that any one of these methods is adopted throughout the Selection GNN, and at each layer ℓ matrix $\mathbf{C}_\ell$ is determined by following the chosen method. On each layer ℓ the subset of nodes selected by $\mathbf{C}_\ell$ is always a subset of the nodes chosen in the previous layer. This implies that $N_\ell \leq N_{\ell-1}$ and that $\mathbf{C}_\ell\mathbf{C}_{\ell-1}$ never yields the zero matrix. In particular, in Sec. V, we adopt the methods proposed in [29] and [32] to study their impact on the overall performance of the Selection GNN.

Locality of filtering. The graph convolution remains a local operation with respect to the original input graph. Since each convolutional feature is zero padded to fit the graph, the implementation of the graph filter at each layer can be carried out by means of local exchanges in the original support. This can be a good computational option if the original input graph is sparse, and therefore repeatedly applying the graph shift operator exploits this sparsity. This turns out to be particularly useful when such a support represents a physical network with physical connections.

Centralized computing. When regarding the selection pooling architecture as a whole, being executed from a single

centralized unit (i.e. when local connectivity is not important for computation purposes, for example, in the training phase), it is observed that the computational cost of carrying out convolutions (13) is reduced to matrix multiplication in the smaller N_ℓ-dimensional space. It is noted that the reduced dimensionality k-shift matrices (12) can be obtained before the training phase, and also, that the statistical properties of learning the filter taps are not affected by it. This observation, coupled with the previous one, shows that the selection pooling architecture adequately addresses the global vs. local duality by efficiently computing convolutions in both settings.

Computation of nonlinearities. From an implementation perspective, it is observed that, while the local summarizing function ρ_ℓ involves the neighborhood of the $N_{\ell-1}$ nodes (which are more than the N_ℓ nodes that are kept in layer ℓ), this function only has to be computed for those N_ℓ nodes that are left after downsampling. That is, it is not needed to compute ρ_ℓ at each one of the $N_{\ell-1}$ nodes, but only at the N_ℓ nodes that are actually kept after downsampling. In this sense, this nonlinear operation can be subsumed with the pointwise nonlinearity σ_ℓ that is applied to the N_ℓ nodes. To further illustrate this point, suppose that max-pooling is used and that the corresponding pointwise nonlinearity is a ReLU, $\sigma_\ell(x) = \max\{0, x\}$. Then, both operations can be performed simultaneously at node n by doing $\max\{0, \{x_m : (m,n) \in \mathbf{n}_\ell\}\}$, where $\mathbf{n}_\ell$ denotes the paths in the neighborhood, and where this operation is computed only for nodes n that are part of the $N_\ell \leq N_{\ell-1}$ selected nodes.

Regularization of filter taps. As the Selection GNN grows in depth (more layers), the number of filter taps in the convolution stage might increase, in order to access information located at further away neighbors (this happens if the few selected nodes at some deeper layer are far away from each other, as measured by the number of neighborhood exchanges). It is a good idea, then, to structure the filter coefficients $\mathbf{h}_\ell^{fg}$ in these deeper layers. More specifically, filtering with N taps might be necessary, so it makes sense to choose $[\mathbf{h}_\ell^{fg}]_k$ constant for a range of k, since no new substantial information is going to be included for a wide range of those k. This reduces the number of trainable parameters and consequently overfitting.

Definition of neighborhoods. Information from the weight function $\mathcal{W}$ of the graph can be included in the pooling stage (15). More precisely, instead of defining the neighborhood only looking at the edge set $\mathcal{E}$, that is $[\mathbf{S}_\ell^{(k)}]_{nm} \neq 0$, we can make $[\mathbf{S}_\ell^{(k)}]_{nm} \geq \delta$ so that we summarize only across edges stronger than δ.

Frequency interpretation of convolutional features. One advantage of having convolutional features defined always on the same graph $\mathcal{G}$ at every layer ℓ is that these can be easily analyzed from a frequency perspective. Since the graph Fourier transform of a signal depends on the eigenvectors $\mathbf{V}$ of the graph shift operator [2], and since the same $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ is used to define all convolutional features [cf. (11)], then they all share the same frequency basis, allowing for a comprehensive frequency analysis at all layers. In particular, if we focus on normal matrix GSOs, i.e. $\mathbf{V}^{-1} = \mathbf{V}^{\mathsf{H}}$, the zero-padding aliasing effect is evidenced in the fact that $\mathbf{V}^{\mathsf{H}}\mathbf{D}^{\mathsf{T}}\mathbf{D}\mathbf{V}$ need not be the identity matrix for arbitrary eigenvectors $\mathbf{V}$ and downsampling matrices $\mathbf{D}$, altering the frequency content of the input signal to a filter. However, the filter taps are learned from the training set, taking into account this aliasing effect, and therefore are able to cope with it, extracting useful features.

Computational cost. The number of computations at each layer is given by the cost of the convolution operation, which is $O(|\mathcal{E}|K_\ell F_\ell F_{\ell-1})$ if (11) is used, or $O(N_{\ell-1}^2 K_\ell F_\ell F_{\ell-1})$ if (13) is used, since pooling and downsampling incur in negligible cost. We observe that in (13) the cost tends to be dominated by $N_{\ell-1}^2$ making dimensionality reduction (i.e. pooling) a critical step for scalability.

Number of parameters. The number of parameters to be learned at each layer are determined by the length of the filters, and the number of input and output features and is given by $O(K_\ell F_\ell F_{\ell-1})$ independent of $N_{\ell-1}$.

IV. Aggregation Graph Neural Networks

The *selection* GNNs of Section III create convolutional features adapted to the structure of the graph with linear shift invariant graph filters. The *aggregation* GNNs that we describe here apply the conventional CNN architecture of Section II to a signal with temporal (regular) structure that is generated to incorporate the topology of the graph. To create such a temporal arrangement we consider successive applications of the graph shift operator $\mathbf{S}$ to the input graph signal $\mathbf{x}^g$ (see first row of Fig. 3). This creates a sequence of N graph shifted signals $\mathbf{y}_0^g, \ldots, \mathbf{y}_{N-1}^g$. The first signal of the sequence is $\mathbf{y}_0^g = \mathbf{x}^g$, the second signal is $\mathbf{y}_1^g = \mathbf{S}\mathbf{x}^g$, and subsequent members of the sequence are recursively obtained as $\mathbf{y}_k^g = \mathbf{S}\mathbf{y}_{k-1}^g = \mathbf{S}^k\mathbf{x}^g$. We observe that each vector $\mathbf{y}_k^g$ incorporates the underlying support by means of multiplication by the graph shift operator $\mathbf{S}$. We arrange the sequence of signals $\mathbf{y}_k^g$ into the matrix representation of the graph signal $\mathbf{x}^g$ that we define as

$$\mathbf{X}^g := [\mathbf{y}_0^g, \mathbf{y}_1^g, ..., \mathbf{y}_{N-1}^g] := [\mathbf{x}^g, \mathbf{S}\mathbf{x}^g, ..., \mathbf{S}^{N-1}\mathbf{x}^g]. \quad (18)$$

The matrix $\mathbf{X}^g$ is a redundant representation of $\mathbf{x}^g$. In fact, for any connected graph any row of $\mathbf{X}^g$ is sufficient to recover $\mathbf{x}^g$ as each row contains N linear combinations of $\mathbf{x}^g$ [7]. We thus note that any such row has successfully incorporated the graph structure included in the powers of the graph shift operator $\mathbf{S}$, without any loss of information. Our goal here is to work at a designated node p with the signal $\mathbf{z}_p^g$ that contains the components of the diffusion sequence $\mathbf{y}_k^g$ that are observed at node p (see second row of Fig. 3). This is simply the pth row of $\mathbf{X}^g$ and leads to the definition

$$\mathbf{z}_p^g := \left[\mathbf{X}^g\right]_p^{\mathsf{T}} = \left[\left[\mathbf{x}^g\right]_p; \left[\mathbf{S}\mathbf{x}^g\right]_p; \ldots; \left[\mathbf{S}^{N-1}\mathbf{x}^g\right]_p\right]. \quad (19)$$

The signal $\mathbf{z}_p^g$ is a local representation at node p that accounts for the topology of the graph in a temporally structured manner. Indeed, since the diffusion sequence $\mathbf{y}_k^g$ is generated from a temporal diffusion process the components of the sequence

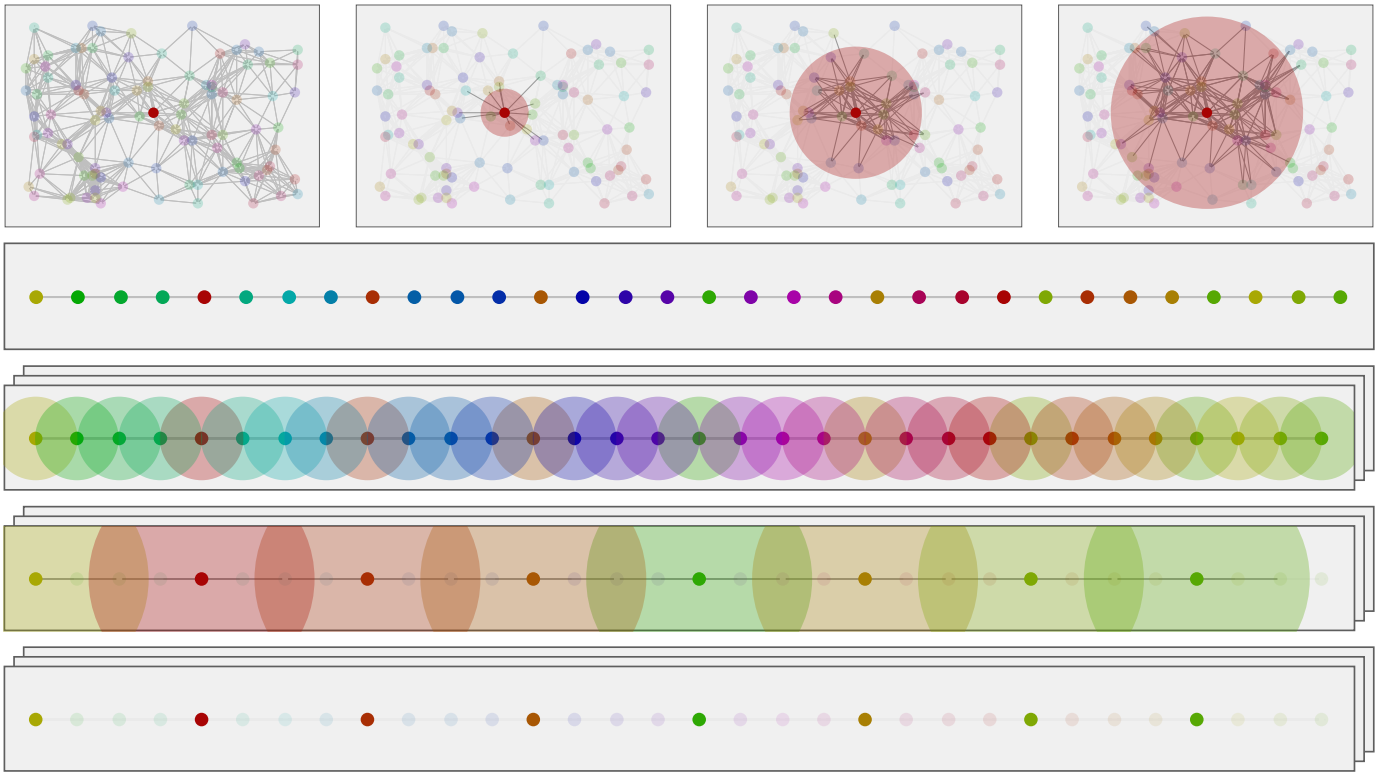


Figure 3. Aggregation Graph Neural Networks. Select a node $p \in \mathcal{V}$ and perform successive local exchanges with its neighbors. For each k-hop neighborhood (illustrated by the increasing disks in the first row), record $\mathbf{S}^k\mathbf{x}^g$ at node p and build signal $\mathbf{z}_p^g$ which exhibits a regular structure [cf. (19)]. Once a regular time-structure signal is obtained, we proceed to apply regular convolution and pooling to process the data [cf. (2)-(5)].

$\mathbf{z}_p^g$ are elements of a time sequence. Yet, the components of this time sequence depend on the topology of the graph. The first element of $\mathbf{z}_p^g$ is the value of the input signal $\mathbf{x}^g$ at node p, which is independent of the graph topology, but the second element $\mathbf{z}_p^g$ aggregates information from values of the input $\mathbf{x}^g$ within the neighborhood of p as defined by the nodes that are connected to node p. The third element of $\mathbf{z}_p^g$ is an aggregate of aggregates which results in the aggregation of information from the 2-hop neighborhood of p. As we move forward in the sequence $\mathbf{z}_p^g$ we incorporate information from nodes that are farther from p as determined by the topology of the graph. In this way, we have successfully generated a regular structured signal that effectively incorporates the underlying structure. We note that two consecutive elements of $\mathbf{z}_p^g$ indeed relate neighboring values according to the topology of the graph.

If the signal $\mathbf{z}_p^g$ is a signal in time, it can be processed with a regular CNN architecture and this is indeed our definition of aggregation GNNs. At the first layer $\ell = 1$ we take the locally aggregated signal $\mathbf{z}_p^g$ as input and produce features $\mathbf{u}_{p1}^{fg}$ by convolving with the K_{p1}-tap filter $\mathbf{h}_{p1}^{fg}$ [cf. (2)],

$$\left[\mathbf{u}_{p1}^{fg}\right]_n := \left[\mathbf{h}_{p1}^{fg} * \mathbf{z}_p^g\right]_n = \sum_{k=0}^{K_{p1}-1} \left[\mathbf{h}_{p1}^{fg}\right]_k \left[\mathbf{z}_p^g\right]_{n-k}, \qquad (20)$$

where we use zero padding to account for border effects and assume the size of the output is the same as the input. The convolution in (20) is the regular time convolution. In fact, except for minor notational differences to identify the aggregation node p, (20) is the same as (2) with $\ell = 1$. The topology of the graph is incorporated in (20) not because of the convolution but because of the way in which we construct $\mathbf{z}_p^g$. To emphasize the effect of the topology of the graph we use (19) to rewrite (20) as

$$\left[\mathbf{u}_{p1}^{fg}\right]_n = \sum_{k=0}^{K_{p1}-1} \left[\mathbf{h}_{p1}^{fg}\right]_k \left[\mathbf{S}^{n-k-1}\mathbf{x}^g\right]_p \qquad (21)$$

Since the convolution in (21) considers consecutive values of the signal $\mathbf{z}_p^g$, the features $\mathbf{u}_{p1}^{fg}$ have a structure that is convolutional on the graph $\mathbf{S}$. Each feature element $[\mathbf{u}_{p1}^{fg}]_n$ is a linear combination of consecutive K_{p1} neighboring values of the input $\mathbf{x}^g$ starting with shift $\mathbf{S}^{n-1}\mathbf{x}^g$ and ending at $\mathbf{S}^{n-K_{p1}-1}\mathbf{x}^g$. Alternatively, note that the regular convolution operation linearly relates consecutive elements of a vector; and since consecutive elements in vector $\mathbf{z}_p^g$ reflect nearby neighborhoods according to the graph, we have effectively related neighboring values of the graph signal by means of a regular convolution. Thus, coefficients $\mathbf{h}_{p1}^{fg}$ encoding low-pass filters further aggregate information across neighborhoods, while high-pass filters output features quantifying differences between consecutive neighborhood resolutions. Thus, low-pass time filters applied to $\mathbf{z}_p^g$ detect features that are smooth on the graph $\mathbf{S}$, while high-pass time filters applied to $\mathbf{z}_p^g$ detect sharp transitions between signal values between nearby nodes.

Once the features $\mathbf{u}_{p1}^{fg}$ in (20), or their equivalents in (21), are computed, we sum features $\mathbf{u}_{p1}^{fg}$ as per (3) obtaining $\mathbf{u}_{p1}^{f}$,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSP.2018.2887403, IEEE Transactions on Signal Processing

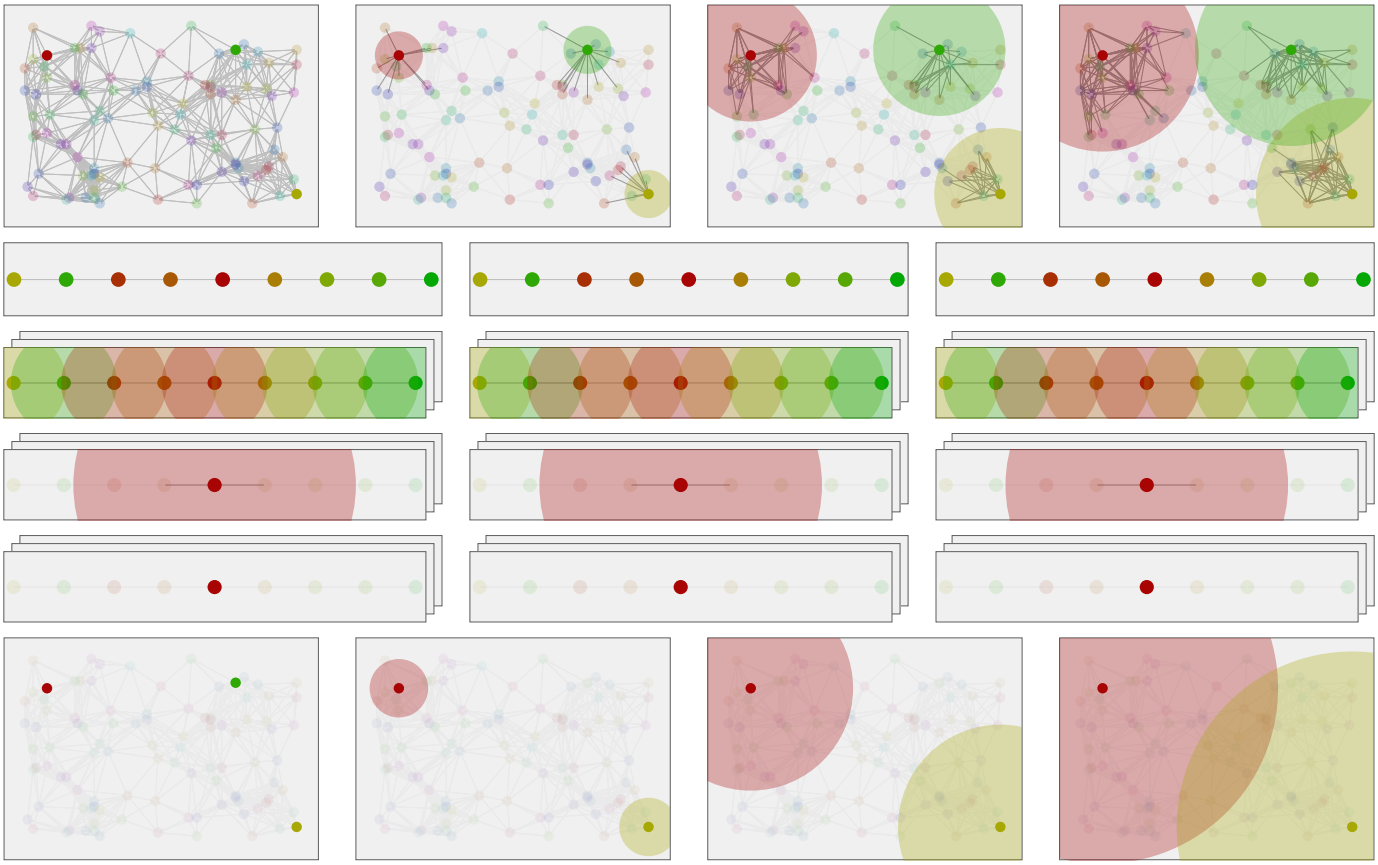


Figure 4. Multinode Aggregation Graph Neural Networks. Start by selecting a subset $\mathcal{P}_1 \subset \mathcal{V}$ of P_1 nodes of the graph (row 1, diagram 1). Then, proceed to perform Q_1 local exchanges with neighbors (row 1, diagrams 2, 3, and 4) in order to build P_1 regular time-structure signals, one at each node (row 2), see (22). We note that in row 1, the color disks illustrate the reach of the Q_1 local exchanges of each of the selected nodes $\mathcal{P}_1$. Once the regular structured signals have been constructed on each of the P_1 nodes, proceed with a regular CNN, applying regular convolution (row 3), and regular pooling (row 4), until F_{L_1} features are obtained at each node (row 5), see (2)-(5), (23). Now, we view each feature as a graph signal being supported on the selected nodes, see (24), zero-padded to fit the graph (row 6, diagram 1), see (25). We then select a smaller subset $\mathcal{P}_2 \subseteq \mathcal{P}_1$ of $P_2 \leq P_1$ nodes (row 2, diagram 2) and carry out Q_2 local exchanges with the neighbors, (row 2, diagrams 2, 3 and 4), illustrated with color disks in the last row. These neighbor exchanges create new regular structured signals at each of the $\mathcal{P}_2$ nodes, see cf. (26). Then, we continue by computing F_{L_2} regional features at each node by means of regular CNNs and so on.

compute local summaries as per (4) yielding $\mathbf{v}_{p1}^f$, and subsample according to (5) resulting in features $\mathbf{x}_{p1}^f$. Since in this case the indexes of the feature vector represent (neighborhood) resolution, some applications may benefit from non-equally spaced sampling schemes that put more emphasis on sampling the high-resolution (low-resolution) part of the feature vector. Subsequent layers repeat the computation of convolutional features and pooling steps in (2)-(5). Formally, all of the variables in (2)-(5) need to be marked with a subindex p to identify the aggregation node.

Remark 4. The aggregation GNN architecture reduces trivially to conventional CNNs when particularized to graph signals defined over a cyclic graph. Since $[\mathbf{A}_{\text{dc}}^k \mathbf{x}^g]_p = [\mathbf{x}^g]_{1+(p+k) \mod N}$ is a cyclic shift of the input signal $\mathbf{x}^g$, then $\mathbf{z}_p^g = \mathbf{x}^g$ in (19) for all p and a regular CNN follows.

Remark 5. The aggregation GNN architecture rests on transforming the data on the graph in such a way that it becomes supported on a regular structure, and thus regular CNN techniques can be applied. Transforming graph data is the main concern of graph embeddings [33]. Unlike the methods surveyed in [33], we consider the underlying graph support $\mathcal{G}$ as given (not learned), we do not attempt to compress the graph data as construction of aggregated vector $\mathbf{z}_p^g$ does not entail any loss of information (if all eigenvalues of $\mathbf{S}$ are distinct), and the focus is on data defined on top of the graph (the graph signal), rather than the graph itself (given by $\mathbf{S}$).

A. Multinode Aggregation Graph Neural Networks

Selecting a single node $p \in \mathcal{V}$ to aggregate all the information generally entails $N-1$ local exchanges with neighbors [cf. (18)]. For large networks, carrying out all these exchanges might be infeasible, either due to the associated communication overhead or due to numerical instabilities. This can be overcome by selecting a subset of nodes to aggregate local information, i.e., selecting a submatrix of (18) with a few rows and columns in lieu of a single row with all the columns; see Fig. 4. The selected nodes will first process their

own samples using an aggregation GNN and then exchange the obtained outputs with the other selected nodes. This process is repeated until the information has been propagated through the entire graph.

To explain such a two-level hierarchical architecture, let us denote as ℓ the layer index for the aggregation stage and as r the layer index for the exchange stage. The total number of exchange (outer) layers is R and, for each outer layer r, a total number of L_r aggregation (inner) layers is run. We start by describing the procedure for $r = 1$, where $\mathcal{P}_1 \subset \mathcal{V}$ denotes the subset of selected nodes and let Q_1 denotes the number of times the shift is applied ($\mathbf{S}^q$, for $q = 0, \ldots, Q_1 - 1$). It is observed that this amounts to selecting $P_1 = |\mathcal{P}_1|$ rows and Q_1 consecutive columns of (18). Setting $\ell = 0$, the signal aggregating the $(Q_1 - 1)$-hop neighborhood information at each node $p \in \mathcal{P}_1$ can be constructed as [cf. (19)]

$$\mathbf{z}^g_{p0}(1, Q_1) := \left[\left[\mathbf{x}^g\right]_p; \left[\mathbf{S}\mathbf{x}^g\right]_p; \ldots; \left[\mathbf{S}^{Q_1-1}\mathbf{x}^g\right]_p \right]. \quad (22)$$

Since $\mathbf{z}^g_{p0}$ exhibits a time structure, the regular CNN steps (2)-(5) can be applied individually at each node (see Fig. 4). More specifically, since L_1 denotes the number of layers for the aggregation stage when $r = 1$, a set of F_{L_1} descriptive features of the $(Q_1 - 1)$-hop neighborhood of node p is constructed by concatenating $\ell = 0, \ldots, L_1 - 1$ layers of the form (2)-(5) as is done in the aggregation GNN. Setting $\ell = L_1$, the output of the last layer of the aggregation stage is

$$\mathbf{z}_{pL_1}(1, Q_1) = \left[z^0_{pL_1}; \ldots; z^{F_{L_1}}_{pL_1} \right]. \quad (23)$$

Different feature vectors $\mathbf{z}_{pL_1}$ of dimension F_{L_1} are obtained at each of the p selected nodes, describing the corresponding $(Q_1 - 1)$-hop neighborhood.

In order to further aggregate these local features (describing local neighborhoods) into more global information, we need to collect each feature g at every selected node $p \in \mathcal{P}_1$. More precisely, let $P_1 = |\mathcal{P}_1|$ be the number of selected nodes, then

$$\mathbf{x}^g_1 = \left[z^g_{p_1 L_1}; \ldots; z^g_{p_{P_1} L_1} \right] \quad (24)$$

where each $p_k \in \mathcal{P}_1$. We now set $r = 2$ and select a subset of nodes $\mathcal{P}_2 \subseteq \mathcal{P}_1$ to aggregate features $\mathbf{x}^g_1$ by means of local neighborhood exchanges. However, signal $\mathbf{x}^g_1$ has dimension $P_1 < N$, so it cannot be directly exchanged through the original graph $\mathcal{G}$. We therefore use zero padding to make $\mathbf{x}^g_1$ fit the graph

$$\tilde{\mathbf{x}}^g_1 = \mathbf{P}_1^{\mathsf{T}} \mathbf{x}^g_1 \quad (25)$$

with $\mathbf{P}_1$ being the $P_1 \times N$ fat binary matrix that selects the subset $\mathcal{P}_1$ of rows of (18). Then, we apply Q_2 times the original shift $\mathbf{S}$ to the signals $\tilde{\mathbf{x}}^g_1$, collecting information at nodes $p \in \mathcal{P}_2$,

$$\mathbf{z}^g_{p0}(2, Q_2) := \left[\left[\tilde{\mathbf{x}}^g_1\right]_p, \left[\mathbf{S}\tilde{\mathbf{x}}^g_1\right]_p, \ldots, \left[\mathbf{S}^{Q_2-1}\tilde{\mathbf{x}}^g_1\right]_p \right]^{\mathsf{T}}. \quad (26)$$

Once $\mathbf{z}^g_{p0}$ is collected at each node $p \in \mathcal{P}_2$ the time-structure of the signal is exploited to deploy another regular CNN (2)-(5) (aggregation GNN stage) in order to obtain F_{L_2} features that describe the region.

In general, consider the output of *outer layer* $r-1$ is $\mathbf{x}^g_{r-1}$, consisting of feature g at a subset $\mathcal{P}_{r-1}$ of P_{r-1} nodes [cf. (24)], for $g = 1, \ldots, F_{L_{r-1}}$. Then, this signal is zero padded to fit the original graph $\tilde{\mathbf{x}}^g_{r-1} = \mathbf{P}^{\mathsf{T}}_{r-1}\mathbf{x}^g_{r-1}$ [cf. (25)] and the graph shift $\mathbf{S}$ is applied Q_r times, collecting the shifted versions at a subset of nodes $\mathcal{P}_r$ to construct time-structure signal $\mathbf{z}^g_{p0}(r, Q_r)$ [cf. (26)]. Each node $p \in \mathcal{P}_r$ runs a regular CNN (2)-(5) with L_r *inner layers* to produce F_{L_r} features $\mathbf{z}_{pL_r}(r, Q_r)$ [cf. (23)] that are then collected at each of the nodes $p \in \mathcal{P}_r$ to produce $\mathbf{x}^f_r$ [cf. (24)], for $f = 1, \ldots, F_{L_r}$. See Fig. 4 for an illustration of the architecture.

B. Practical Considerations

Local architecture. The single node aggregation GNN architecture is entirely *local*. Only one node $p \in \mathcal{V}$ is selected, and that node gathers all the relevant information about the data by means of local exchanges only. Furthermore, the output at the last layer is also obtained at a single node, so there is no need to have actual physical access to every node in the network.

Regular CNN design. Since signal $\mathbf{z}^g_p$ gathered at node p exhibits a regular time structure, the state-of-the-art expertise in designing conventional CNNs can be immediately leveraged to inform the design of convolutional layers of aggregation GNNs.

Numerical normalization. For big networks, some of the entries of $\mathbf{S}^k$ (as well as the components of $\mathbf{z}^g_p$ associated with those powers) can grow too large, leading to numerical instability. To avoid this, aggregation schemes typically work with a normalized version of the graph shift operator that guarantees that the spectral radius of $\mathbf{S}$ is one.

Choice of aggregating node. The choice of nodes that aggregate all the information has an impact on the overall performance of the algorithm. This decision can be informed by several criteria such as the degree, the frequency content of the signals of interest [7] or be determined by different measures of centrality in the network [34]. In particular, in the experiments carried out in Sec. V, we select nodes based on the leverage scores obtained by the two sampling schemes described in [29] and [32].

Filter taps. For a generic (inner) layer $1 < \ell < L_r$ the generation of the feature vectors $\mathbf{u}^{fg}_\ell \in \mathbb{R}^{N_{\ell-1}}$ and $\mathbf{u}^f_\ell \in \mathbb{R}^{N_{\ell-1}}$ is as in (2) and (3), so that we have that $\mathbf{u}^f_\ell = \sum_{g=1}^{F_{\ell-1}} \mathbf{u}^{fg}_\ell = \sum_{g=1}^{F_{\ell-1}} \mathbf{h}^{fg}_\ell * \mathbf{z}^g_{p(\ell-1)}$. The main difference in this case is on the type and length of the filter coefficients $\mathbf{h}^{fg}_\ell \in \mathbb{R}^{K_\ell}$. While in classical CNNs the filter coefficients are critical to aggregate the information at different resolutions, here part of that aggregation has been already taken care of in the first layer when transforming $\mathbf{x}^g$ into $\mathbf{z}^g_p$. As a result, the filter taps in the aggregation GNN architecture can have a shorter length and place more emphasis in high frequency features.

Pooling. Something similar applies to the pooling schemes. The summarization and downsampled vectors for the aggregation architecture are obtained as $[\mathbf{v}^f_\ell]_n = \rho_\ell([\mathbf{u}^f_\ell]_{\mathbf{n}_\ell})$ and

$\mathbf{x}_\ell^f = \sigma_\ell(\mathbf{C}_\ell \mathbf{v}_\ell^f)$, which coincide with their counterparts for classical CNNs in (4) and (5). The difference is therefore not in the expressions, but on how $\mathbf{n}_\ell$ and $\mathbf{C}_\ell$ are selected. While in traditional CNNs the signal $\mathbf{x}^g$ is global in that all the samples have the same resolution, in the aggregation architecture the signal $\mathbf{z}_p^g$ is local and different samples correspond to different levels of resolution. More specifically, aggregation pooling schemes for $\mathbf{n}_\ell$ and $\mathbf{C}_\ell$ that preserve the top samples of the feature vectors $\mathbf{u}_\ell^f$ to keep finer resolutions combined with a few bottom samples to account for coarser information are reasonable, while in traditional CNNs regular schemes for $\mathbf{n}_\ell$ and $\mathbf{C}_\ell$ that extract information and sample the signal support regularly can be more adequate.

Design flexibility. The multinode aggregation GNN acts as a decentralized method for constructing regional features. We note that, for ease of exposition, the number of shifts Q_r at each outer layer is the same for all nodes as well as the number of features F_{L_r} that are obtained at each node. However, this architecture can be adapted to different node-dependent parameters in a straightforward manner.

Computational cost. The computational cost of the multinode aggregation GNN at each outer layer r is that of processing the regular CNN for each node, $O(\sum_{p=1}^{P_r} \sum_{\ell=1}^{L_r} N_{\ell-1} K_\ell F_{\ell-1} F_\ell)$ which can be easily distributed among the P_r involved nodes.

Number of parameters. The number of parameters of the multinode aggregation GNN is $O(\sum_{p=1}^{P_r} \sum_{\ell=1}^{L_r} K_\ell F_\ell F_{\ell-1})$. We observe, though, that the regular CNNs employed tend to be very small, since the initial Q_r regular CNN at each node) as well as the length of the filters K_ℓ are very small as well (typically, $K_\ell \ll Q_r$, cf. Sec. II).

V. Numerical Experiments

We test the proposed GNN architectures and compare their performance with the graph coarsening (multiscale hierarchical clustering) approach of [16]. In the first scenario (Sec. V-A), we address the problem of source localization on synthetic stochastic block model (SBM) networks. Then, we move the source localization problem to a more realistic setting of a Facebook network of 234 users (Sec. V-B). As a third experiment, we address the problem of authorship attribution (Sec. V-C). And finally, we test the proposed architectures in the problem of text categorization on the `20NEWS` dataset (Sec. V-D).

We test the proposed Selection (Sec. III), Aggregation (Sec. IV) and Multinode (Sec. IV-A) GNN architectures. For the selection of nodes involved in each of the architectures, we test three different strategies. First, we choose nodes based on their degree; second, we select them following the leverage scores proposed by the experimentally designed sampling (EDS) in [32]; and third, we determine the appropriate nodes by using the spectral-proxies approach (SP) in [29]. In all architectures, the last layer is a fully-connected readout layer, followed by a softmax, to perform classification.

Unless otherwise specified, all GNNs were trained using the ADAM optimizer [35] with learning rate 0.001 and forgetting factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The training phase is carried out for 40 epochs with batches of 100 training samples. The loss function considered in all cases is the cross-entropy loss between one-hot target vectors and the output from the last layer of each architecture, interpreted as probabilities of belonging to each class. Also, in all cases, we consider max-pooling summarizing functions and ReLU activation functions for the corresponding GNN layers.

Architecture	Accuracy
Selection (S) Degree	$86.9(\pm 5.9)\%$
Selection (S) EDS	$90.0(\pm 4.6)\%$
Selection (S) SP	$91.1(\pm 4.7)\%$
Aggregation (A) Degree	$94.2(\pm 4.7)\%$
Aggregation (A) EDS	$96.5(\pm 3.1)\%$
Aggregation (A) SP	$95.2(\pm 4.4)\%$
Multinode (MN) Degree	$96.1(\pm 3.4)\%$
Multinode (MN) EDS	$96.0(\pm 3.5)\%$
Multinode (MN) SP	$\mathbf{97.3}(\pm \mathbf{2.7})\%$
Graph Coarsening (C) Clustering	$87.4(\pm 3.2)\%$

Table I: Considering that SBM graphs are random, we generate 10 different instances of SBM graphs with $N = 100$ nodes and $C = 5$ communities of 20 nodes each. For each of these 10 graphs, we randomly generate 10 different datasets (training, validation and test set). We compute the classification accuracy of each realization, and average across all 10 realizations for each graph, obtaining 10 average classification accuracies. In the table we show the classification accuracy, averaged across the 10 graph instances. The standard deviation from these 10 graphs is also shown.

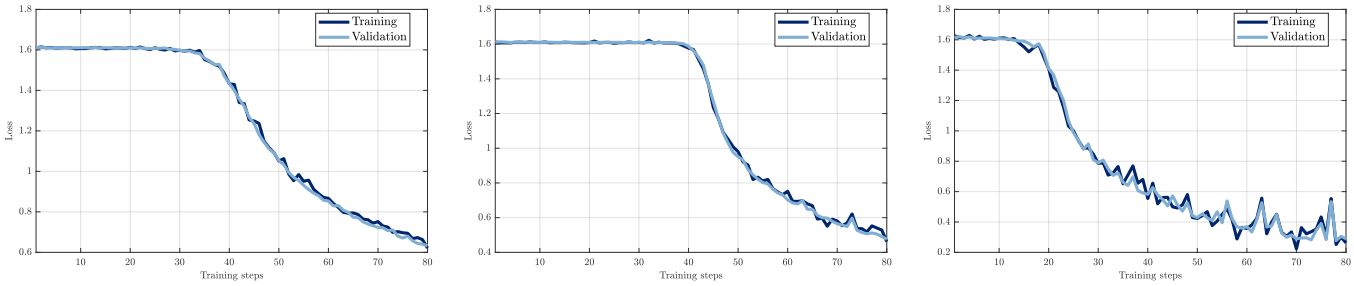
A. Source Localization

Consider a connected stochastic block model (SBM) network with $N = 100$ nodes and $C = 5$ communities of 20 nodes each [36]. In SBM graphs, edges are randomly drawn between nodes within the same community, independently, with probability 0.8; while edges are randomly drawn between nodes of different communities, independently, with probability 0.2. Denote by $\mathbf{A}$ the adjacency matrix of such graph.

In the problem of source localization, we observe a signal that has been diffused over the graph and estimate the spatial origin of such diffused process. More precisely, consider $\boldsymbol{\delta}_c$ a graph signal that has a 1 at node c and 0 at every other node. Define $\mathbf{x} = \mathbf{A}^t \boldsymbol{\delta}_c$ as the diffused graph signal, for some unknown $t \geq 0$. The objective is to estimate the origin c of the diffusion. In this situation in particular, we are interested in estimating the *community* c (rather than the node) that originated the observed signal $\mathbf{x}$. We can thus model this scenario as a classification problem in which we observe graph signal $\mathbf{x}$ and have to assign it to one of the $C = 5$ communities.

In the simulations, we generate the training and test set by randomly selecting the origin c from a pool of $C = 5$ nodes (the largest-degree node of each community; recall that all nodes have, on average, the same degree) and randomly selecting the diffusion time $t < 25$, as well. We generate a training set of $10,000$ signals and a test set of 200 signals. The training set is further split in $2,000$ signals for validation, and the rest for training. We simulate 10 graphs, and for each graph, we simulate 10 realizations of training and test sets.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSP.2018.2887403, IEEE Transactions on Signal Processing



(a) Selection GNN SP (b) Aggregation GNN EDS (c) Multinode GNN SP

Figure 5. Validation and training loss during training stage. We observe that the validation loss and the training loss are essentially equal throughout the training stage for all three architectures. This shows that the proposed models are not overfitting the data, since the validation loss keeps decreasing with the training steps The best performing selection method of each architecture is represented.

For numerical reasons, the adopted graph shift operator is $\mathbf{S} = \mathbf{A}/\lambda_{\max}$ where $\lambda_{\max}$ is the maximum eigenvalue of $\mathbf{A}$.

The architectures tested are as follows. For the selection GNN we consider two layers selecting 10 nodes in each. The number of output features in each layer are $F_1 = F_2 = 32$ and the filters consists of $K_1 = K_2 = 5$ taps [cf. (13)]. For the summarizing functions, we consider neighborhoods of size $\alpha_1 = 6$ and $\alpha_2 = 8$, respectively [cf. (15)]. In the aggregation GNN, we select the single node with highest: a) degree, b) EDS leverage score, or d) spectral-proxies (SP) norm, depending on the strategy chosen. Then, we construct the regular-structured signal [cf. (19)] and apply the aggregation GNN with two layers. The number of features on each layer is $F_1 = 16$ and $F_2 = 32$, with filters of size $K_1 = 4$ and $K_2 = 8$ [cf. (21)]. Max-pooling is applied to reduce the size of the regular signal by half on each layer, and the nonlinearity used is the ReLU. Finally, for the multinode GNN, we consider two outer layers selecting $P_1 = 10$ and $P_2 = 5$ nodes and shifting the signal $Q_1 = 7$ and $Q_2 = 5$ times to build the regular signal on each node [cf. (22)]. Then, for each outer layer, we apply two inner layers. In the first one, we obtain 16 features at each inner layer; and in the second outer layer, we get 16 and 32 for each inner layer. In all inner layers, the filters are of size 3, with max-pooling by 2, and a ReLU nonlinearity. We recall that the selection of nodes depends on the sampling strategy selected (degree, EDS or SP). We compare against a two-layer architecture using graph coarsening [16], reducing the number of nodes to a half on each layer, computing $F_1 = F_2 = 32$ features with filters consisting of $K_1 = K_2 = 5$ filter taps. In contrast with the previous cases where $\mathbf{S}$ was set to the rescaled adjancency matrix, in the graph coarsening architecture we set $\mathbf{S}$ to normalized Laplacian, since that was the specification in [16] and, more importantly, yields a better performance.

The plots in Fig. 5 show the value of the loss function on the training and validation sets as the training stage progresses. We observe that both drop with training, showing that the model is effectively learning from data. We see that it takes some time for the models to start learning (reaching half of the training stage in the case of aggregation), but then effectively lower the training loss. We also see that the Multinode GNN achieves a lower loss value, which translates in better performance on the test set, and that it also takes the least number of training steps before starting to lower the loss function. Finally, we note that the validation loss and the training loss are essentially the same, showing that there is no overfit in the models.

Accuracy results on the test set are presented in Table I. The accuracy results for all 10 realizations of each graph are averaged, and then all 10 graph mean accuracies are averaged to obtain the values shown in Table I. The error values in the table are the square root of the variance computed across the means obtained for each of the 10 graphs. We observe that the best performance is achieved by Multinode GNN with nodes chosen following the spectral proxies method. We observe that all multinode and aggregation GNNs outperform the graph coarsening approach, and so do selection GNNs following EDS and spectral proxies sampling.

B. *Facebook network*

For this second experiment, we also consider the source localization problem, but in this case, we test it on top of a real-world network. In particular, we built a 234-user Facebook network as the largest connected network within the larger dataset provided in [37]. We observe that the resulting network exhibits two communities of quite different size. The source localization problem formulation is the same than the one described in the previous section, where the objective is to identify which of the two communities originated the diffusion process. This is analogous to finding the start of a rumor. Again, we set $\mathbf{S} = \mathbf{A}/\lambda_{\max}$. The datasets are generated in the same fashion as described in the previous section.

The three architectures used are as follows. For the selection GNN we use two layers, choosing 10 nodes after the first one, and use filters with $K_1 = K_2 = 5$ taps that generate $F_1 = F_2 = 32$ features on each layer. For the pooling stage, we use a $\max\{\cdot\}$ summarization with $\alpha_1 = 2$ and $\alpha_2 = 4$. In the aggregation GNN we select the best node based on one of the three sampling strategies (degree, EDS and SP) and the gather the regular-structure data at that node. We then process it with a two-layer CNN that generates $F_1 = 32$ and $F_2 = 64$ features, using $K_1 = K_2 = 4$. Max-pooling of size 2 is used on each layer (i.e. half of the samples gathered at the node are kept after each layer). In the case of the multinode GNN we use two-outer layers, selecting $P_1 = 30$ and $P_2 = 10$ nodes on each, and gathering $Q_1 = Q_2 = 5$ shifted versions

Architecture	Accuracy
Selection (S) Degree	$96.0(\pm 1.5)\%$
Selection (S) EDS	$95.6(\pm 1.0)\%$
Selection (S) SP	$97.6(\pm 1.2)\%$
Aggregation (A) Degree	$95.8(\pm 1.6)\%$
Aggregation (A) EDS	$96.9(\pm 1.2)\%$
Aggregation (A) SP	$95.8(\pm 1.4)\%$
Multinode (MN) Degree	$97.6(\pm 1.3)\%$
Multinode (MN) EDS	$96.8(\pm 1.2)\%$
Multinode (MN) SP	$\mathbf{99.0}(\pm \mathbf{0.8})\%$
Graph Coarsening (C) Clustering	$95.2(\pm 1.2)\%$

Table II: Classification accuracy averaged across 10 different realizations of the training and test sets for the same underlying graph. In parenthesis, we show the standard deviation of the classification accuracy.

Architecture	Accuracy
Selection (S) Degree	$69.6(\pm 5.6)\%$
Selection (S) EDS	$68.1(\pm 5.3)\%$
Selection (S) SP	$73.0(\pm 4.8)\%$
Aggregation (A) Degree	$69.5(\pm 2.0)\%$
Aggregation (A) EDS	$71.0(\pm 2.8)\%$
Aggregation (A) SP	$69.2(\pm 4.0)\%$
Multinode (MN) Degree	$80.4(\pm 2.0)\%$
Multinode (MN) EDS	$\mathbf{80.5}(\pm \mathbf{2.6})\%$
Multinode (MN) SP	$79.9(\pm 2.8)\%$
Graph Coarsening (C) Clustering	$65.2(\pm 5.0)\%$

Table III: Classification accuracy averaged across 10 different realizations of the training and test sets (recall that the training and test sets are chosen at random from the available corpus, and the choice of training set affects the constructed underlying graph). In parenthesis, we show the standard deviation of the classification accuracy.

of the signal at each node. Then, for the inner layers, we use two-layer architectures that generate 16 features on each layer in the first outer layer, and 16 and 32 features on each layer in the second outer layer. In all cases, we use filters of size 3 and max-pooling by a factor of 2. Finally, for the graph coarsening architecture, we adopt the normalized Laplacian as GSO, as described in [16], and use two-layers computing $F_1 = F_2 = 32$ features using graph filters with $K_1 = K_2 = 5$ filter taps. After each layer, the number of nodes is reduced by half.

For training we use 80 epochs. We also generate 10 different random realizations of the dataset to account for random variabilities in the setting. Results for all ten architectures are shown in Table II. We observe that all architectures achieve a very high classification accuracy. We note that selection GNN tends to outperform aggregation GNN. The best result is obtained for multinode GNN using spectral proxies and is 99.0% classification accuracy.

C. Authorship attribution

As a third experiment, we study the problem of authorship attribution, as detailed in [38]. We consider excerpts of works written by a myriad of contemporary authors from the 19th century. We then build a word adjacency network (WAN) using functional words in these excerpts, and obtain a graph profile for each author, i.e., a graph that represents an author's writing style by the way functional words (who act as nodes) are linked (weighted edges) in the excerpts written; see [38] for a full detail on the authors considered and the specific construction of WANs. Then, we take a new excerpt, of unknown authorship, and by looking at the frequency of the functional words, we want to determine who the author is. In the framework presented in this paper, the signature word adjacency network constitutes the underlying graph support, and the frequency count of functional words becomes the graph signal.

In particular, we focus on texts authored by Emily Brontë. We consider a corpus of 682 excerpts of around 1000 words, authored by her; and take into consideration 211 functional words. Then, we take 546 of these excerpts as a training set, in order to both, build the signature WAN, and also as training samples. The constructed graph consists of $N = 211$ nodes, one for each functional word, the edges and their weights are determined by the precedence relationship between each word, as described in [38]; and each training sample consist of a graph signal, where the value associated to each node is the frequency count of that specific functional word. The remaining 136 excerpts are used as test samples. Once the signature WAN for Brontë is built, we construct a training set of 1092 text excerpts, 546 corresponding to the author, and 546 corresponding to other contemporary authors; and a test set of 272 excerpts, 136 belonging to Brontë, and 136 written by other authors. The excerpts corresponding to the training and test set, written by either Brontë or other contemporary authors, are chosen uniformly at random. The objective is to decide if the excerpts in the test set were written by Brontë.

Again, we consider the three GNN architectures proposed in this paper, as well as the graph coarsening GNN of [16]. For the selection GNN, we consider a two-layer architecture, where we choose 100 nodes (functional words) as determined by each of the three sampling strategies (degree, EDS and SP). For each layer we set $F_1 = F_2 = 32$, $K_1 = K_2 = 5$ and $\alpha_1 = 2$ and $\alpha_2 = 4$. In the aggregation GNN we consider three layers, after aggregating all the information at the chosen node (the choice depends on each sampling strategy). In the first layer we compute $F_1 = 32$ features with a filter of size $K_1 = 6$, and do max-pooling, reducing the number of samples by 4. The second and third layers use filters of size $K_2 = K_3 = 4$ to obtain $F_2 = 64$ and $F_3 = 128$ features respectively. Pooling is applied, reducing the size of the vector by a factor of 2 in each of the last two aggregation GNN layers. The multinode GNN employed consists of two outer layers, choosing $P_1 = 30$ and $P_2 = 10$ nodes, respectively, and aggregating information up to the $Q_1 = Q_2 = 5$ hop-neighborhood. For each outer layer, we have two inner layers, having 16 features on each of those for the first outer layer, and 16 and 32 features for the second outer layer. All filters are of size 3 and pooling reduces the size of the vectors by half. Finally, the graph coarsening GNN consists of two layers obtaining $F_1 = F_2 = 32$ features in each, with graph filters of size $K_1 = K_2 = 5$, and pooling reducing the size of the graph by half on each layer.

The graph shift operator $\mathbf{S}$ is set to the adjacency matrix after normalizing the weights of each row (to add up to 1) and symmetrizing it, except for the case of graph coarsening

Architecture	Accuracy
Selection (S) Degree	$55.7(\pm 0.5)\%$
Selection (S) EDS	$58.1(\pm 0.5)\%$
Selection (S) SP	$59.2(\pm 0.4)\%$
Aggregation (A) Degree	$49.0(\pm 0.4)\%$
Aggregation (A) EDS	$51.3(\pm 0.5)\%$
Aggregation (A) SP	$52.9(\pm 0.5)\%$
Multinode (MN) Degree	$65.7(\pm 0.4)\%$
Multinode (MN) EDS	$66.5(\pm 0.5)\%$
Multinode (MN) SP	$\mathbf{67.0}(\pm \mathbf{0.5})\%$
Graph Coarsening (C) Clustering	$62.8(\pm 0.5)\%$

Table IV: 20NEWS dataset on a word2vec graph embedding of $N = 1,000$ nodes. Classification accuracy averaged across 10 different runs. In parenthesis, we show the standard deviation of the classification accuracy.

GNNs, where the GSO is the normalized Laplacian obtained from the aforementioned adjacency matrix. For training we use 80 epochs. And we run the experiment 10 times, to account for the randomness in the selection of training and test sets (and thus, for the randomness in the creation of the underlying WAN).

Results can be found in Table III, where we show the classification accuracy averaged over 10 different realizations of the training and test sets, as well as the estimated standard deviation. We first observe that, in this case, all proposed GNN architectures outperform the graph coarsening GNN. We note that the multinode GNN is the best performing architecture. We also observe that selecting nodes via the EDS sampling method works best for aggregation and multinode GNNs, but spectral proxies yield better results in the case of selection GNN. The best classification accuracy obtained is 80.5%, on average across all realizations, and achieved by the multinode GNN whose nodes are selected by means of EDS sampling.

D. *20NEWS dataset*

Finally, we consider the classification of articles in the 20NEWS dataset which consists of $16,617$ texts ($9,922$ of which are used for training and $6,695$ for testing) [39]. The graph signals are constructed as in [16]: each document x is represented using a normalized bag-of-words model and the underlying graph support is constructed using a 16-NN graph on the word2vec embedding [40] considering the $1,000$ most common words. The GSO adopted is the normalized Laplacian as in [16].

The selection GNN architecture consists of 2 convolutional layers, selecting $P_1 = 250$ and $P_2 = 100$ nodes, according to each of the three different sampling strategies. Each layer uses graph filters of $K_1 = K_2 = 5$ taps to build $F_1 = 32$ and $F_2 = 64$ features. The pooling neighborhoods correspond to $\alpha_1 = 7$ and $\alpha_2 = 12$. For the aggregation GNN we also consider 2 layers, and use filters of length $K_1 = K_2 = 11$ to build $F_1 = F_2 = 32$ features on each layer. Pooling size is 4, and the data is aggregated at a single node chosen by each of the sampling strategies. The multinode GNN consists of 2 outer layers that select $P_1 = 70$ and $P_2 = 30$ nodes, respectively. The number of local exchanges to create a temporally-structured signal are $Q_1 = 10$ and $Q_2 = 25$. Each outer layer employs a regular CNN with 2 inner layers. Each inner layer of the first outer layer creates 16 features, while each inner layer of the second outer layer uses 16 and 32 features, respectively. All filters involved are of length 5 and the pooling size is 4. Finally, for the graph coarsening architecture, we consider 2 layers, reducing the number of nodes by half on each layer, and computing $F_1 = 32$ and $F_2 = 64$ features, using filters of length $K_1 = K_2 = 5$.

Training is done for 80 epochs. Classification accuracy results, averaged out of 10 runs, are listed in Table IV. We note that the multinode GNN is the best performing architecture, followed by graph coarsening. The comparatively poor performance of the aggregation GNN is most likely due to the numerical instabilities that arise from performing a large number of neighborhood exchanges.

VI. Conclusions

In this paper we proposed two architectures for extending convolutional neural networks to process graph signals. The selection graph neural network replaces the convolution operation with graph filtering by means of linear shift invariant graph filters. Pooling is reinterpreted as a neighborhood summarizing function that gathers the relevant regional information at a subset of nodes, followed by a downsampling. By keeping track of the location of these subsets of nodes in the original graph, convolutional layers can be further computed at deeper layers through the use of zero padding. In this way, the selection GNN respects the original topology that describes the data, while reducing the computational complexity at each layer. Furthermore, the resultant features at each layer can be appropriately analyzed in terms of the original graph (frequency analysis, local filtering).

The aggregation GNN collects, at a single node, diffused versions of the original signal. The resulting signal simultaneously possesses a regular temporal structure and includes all relevant information of the topology of the graph. Since the signal collected at this single node has a temporal structure, a regular CNN can be applied to it. In large scale networks, however, gathering all the information of the graph signal at a single node might be infeasible. In order to overcome this, we proposed a multinode variation of the aggregation GNN in which we use a subset of nodes to subsequently create meaningful features of increasing neighborhoods.

We have tested the proposed architectures in a source localization problem on both synthetic and real datasets, as well as for authorship attribution and the classification of articles of the 20NEWS dataset. We considered three different ways of choosing nodes in each architecture, based on three existing sampling techniques (namely, by degree, and by leverage scores computed from experimentally designed sampling and spectral proxies). We compared the results with an existing graph coarsening GNN that employs multiscale hierarchical clustering for the pooling stage. We observe that the multinode aggregation GNN exhibits the best performance.

All in all, the proposed GNN architectures exploit the advances in graph signal processing to present novel constructions of deep learning that are able to handle network data represented as signals supported on graphs.

REFERENCES

[1] 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.

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

[3] 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.

[4] A. Sandryhaila and J. M. F. Moura, "Big data analysis with signal processing on graphs," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 80–90, Sep. 2014.

[5] 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.

[6] D. I. Shuman, P. Vandergheynst, D. Kressner, and P. Frossard, "Distributed signal processing via chebyshev polynomial approximation," *IEEE Trans. Signal, Inform. Process. Netw.*, 6 Apr. 2018, early access.

[7] A. G. Marques, S. Segarra, G. Leus, and A. Ribeiro, "Sampling of graph signals with successive local aggregations," *IEEE Trans. Signal Process.*, vol. 64, no. 7, pp. 1832–1843, Apr. 2016.

[8] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, and N. Seliya, "Deep learning applications and challenges in big data analytics," *J. Big Data*, vol. 2, no. 1, pp. 1–21, Dec. 2015.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 85–117, 2015.

[11] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *2010 IEEE Int. Symp. Circuits and Syst.* Paris, France: IEEE, 30 May-2 June 2010.

[12] H. Greenspan, B. van Ginneken, and R. M. Summers, "Deep learning in medical imaging: Overview and future promise of an exciting new technique," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1153–1159, May 2016.

[13] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," *arXiv:1312.6203v3 [cs.LG]*, 21 May 2014. [Online]. Available: http://arxiv.org/abs/1213.6203

[14] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv:1506.051631v1 [cs.LG]*, 16 June 2015. [Online]. Available: http://arxiv.org/abs/1506.05163

[15] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *30th Neural Inform. Process. Syst.* Barcelona, Spain: NIPS Foundation, 5-10 Dec. 2016.

[16] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Neural Inform. Process. Syst. 2016*. Barcelona, Spain: NIPS Foundation, 5-10 Dec. 2016.

[17] J. Du, S. Zhang, G. Wu, J. M. F. Moura, and S. Kar, "Topology adaptive graph convolutional networks," *arXiv:1710.10370v2 [cs.LG]*, 2 Nov. 2017. [Online]. Available: http://arxiv.org/abs/1710.10370v2

[18] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th Int. Conf. Learning Representations*. Toulon, France: Assoc. Comput. Linguistics, 24-26 Apr. 2017.

[19] F. Gama, A. G. Marques, A. Ribeiro, and G. Leus, "MIMO graph filters for convolutional networks," *arXiv:1803.02247v1 [cs.LG]*, 6 March 2018. [Online]. Available: http://arxiv.org/abs/1803.02247

[20] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *33rd Int. Conf. Mach. Learning*, New York, NY, 24-26 June 2016.

[21] B. Pasdeloup, V. Gripon, J.-C. Vialatte, and D. Pastor, "Convolutional neural networks on irregular domains through approximate translations on inferred graphs," *arXiv:1710.10035v1 [cs.DM]*, 27 Oct. 2017. [Online]. Available: http://arxiv.org/abs/1710.10035

[22] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *arXiv:1710.10903v3 [stat.ML]*, 4 Feb. 2018. [Online]. Available: http://arxiv.org/abs/1710.10903

[23] F. Gama, G. Leus, A. G. Marques, and A. Ribeiro, "Convolutional neural networks via node-varying graph filters," in *2018 IEEE Data Sci. Workshop*. Lausanne, Switzerland: IEEE, 4-6 June 2018.

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

[25] C.-C. J. Kuo, "The CNN as a guided multilayer RECOS transform," *IEEE Signal Process. Mag.*, vol. 34, no. 3, pp. 81–89, May 2017.

[26] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recognition 2017*. Honolulu, HI: IEEE Comput. Soc., 21-26 July 2017.

[27] T. Wiatoski and H. Bölcskei, "A mathematical theory of deep convolutional neural networks for feature extraction," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1845–1866, March 2018.

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

[29] A. Anis, A. Gadde, and A. Ortega, "Efficient sampling set selection for bandlimited graph signals using graph spectral proxies," *IEEE Trans. Signal Process.*, vol. 64, no. 14, pp. 3775–3789, July 2016.

[30] M. Tsitsvero, S. Barbarossa, and P. Di Lorenzo, "Signals on graphs: Uncertainty principle and sampling," *IEEE Trans. Signal Process.*, vol. 64, no. 18, pp. 4845–4860.

[31] G. Puy, N. Tremblay, R. Gribonval, and P. Vandergheynst, "Random sampling of bandlimited graph signals," *Appl. Comput. Harmonic Anal.*, vol. 44, no. 2, pp. 446–475, March 2018.

[32] R. Varma, S. Chen, and J. Kovačević, "Spectrum-blind signal recovery on graphs," in *2015 IEEE Int. Workshop Comput. Advances Multi-Sensor Adaptive Process.* Cancún, México: IEEE, 13-16 Dec. 2015, pp. 81–84.

[33] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques and applications," *arXiv:1709.07604v3 [cs.AI]*, 2 Feb. 2018. [Online]. Available: http://arxiv.org/abs/1709.07604

[34] S. Segarra and A. Ribeiro, "Stability and continuity of centrality measures in weighted graphs," *IEEE Trans. Signal Process.*, vol. 64, no. 3, pp. 543–555, Feb. 2016.

[35] D. P. Kingma and J. L. Ba, "ADAM: A method for stochastic optimization," in *3rd Int. Conf. Learning Representations*. San Diego, CA: Assoc. Comput. Linguistics, 7-9 May 2015.

[36] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborová, "Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications," *Physical Review E*, vol. 84, no. 6, p. 066106, Dec. 2011.

[37] J. McAuley and J. Leskovec, "Learning to discover social circles in Ego networks," in *26th Neural Inform. Process. Syst.* Stateline, TX: NIPS Foundation, 3-8 Dec. 2012.

[38] S. Segarra, M. Eisen, and A. Ribeiro, "Authorship attribution through function word adjacency networks," *IEEE Trans. Signal Process.*, vol. 63, no. 20, pp. 5464–5478, Oct. 2015.

[39] T. Joachims, "Analysis of the rocchio algorithm with tfidf for text categorization," Carnegie Mellon University, Computer Science Technical Report CMU-CS-96-118, 1996.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st Int. Conf. Learning Representations*. Scottsdale, AZ: Assoc. Comput. Linguistics, 2-4 May 2013.