Convolutional Graph Neural Networks

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

DOI
10.1109/IEEECONF44664.2019.9048767

Publication date
2019

Document Version
Final published version

Published in
Conference Record - 53rd Asilomar Conference on Circuits, Systems and Computers, ACSSC 2019

Convolutional Graph Neural Networks

Fernando Gama
Dept. of Electrical and Systems Eng.
University of Pennsylvania
Philadelphia, USA
fgama@seas.upenn.edu

Antonio G. Marques
Dept. of Signal Theory and Comms.
King Juan Carlos Univ.
Madrid, Spain
antonio.garcia.marques@urjc.es

Geert Leus
Circuits and Systems Group
Delft Univ. of Technology
Delft, the Netherlands
g.j.t.leus@tudelft.nl

Alejandro Ribeiro
Dept. of Electrical and Systems Eng.
University of Pennsylvania
Philadelphia, USA
aribeiro@seas.upenn.edu

***Abstract*—Convolutional neural networks (CNNs) restrict the, otherwise arbitrary, linear operation of neural networks to be a convolution with a bank of learned filters. This makes them suitable for learning tasks based on data that exhibit the regular structure of time signals and images. The use of convolutions, however, makes them unsuitable for processing data that do not exhibit such a regular structure. Graph signal processing (GSP) has emerged as a powerful alternative to process signals whose irregular structure can be described by a graph. Central to GSP is the notion of graph convolutional filters which can be used to define convolutional graph neural networks (GNNs). In this paper, we show that the graph convolution can be interpreted as either a diffusion or aggregation operation. When combined with nonlinear processing, these different interpretations lead to different generalizations which we term selection and aggregation GNNs. The selection GNN relies on linear combinations of signal diffusions at different resolutions combined with node-wise nonlinearities. The aggregation GNN relies on linear combinations of neighborhood averages of different depth. Instead of node-wise nonlinearities, the nonlinearity in aggregation GNNs is pointwise on the different aggregation levels. Both of these models particularize to regular CNNs when applied to time signals but are different when applied to arbitrary graphs. Numerical evaluations show different levels of performance for selection and aggregation GNNs.**

***Index Terms*—graph neural networks, graph convolutions, graph signal processing, network data**

I. Introduction

Neural networks are information processing architectures consisting of a cascade of operational *layers*, each one applying a linear projection followed by an activation function (typically, a pointwise nonlinearity). Neural networks are used as nonlinear parameterizations of mappings between the input data and some desired target representation that is relevant for the task at hand. The linear transforms are optimized to *fit* some available training dataset. The chosen cost function to minimize, not only has to be differentiable to allow for gradient descent algorithms, but also has to yield a neural network with good *generalization* properties. This means that the output of the *trained* neural network on an unseen point (not belonging to the training set), still leads to a good estimate of the target representation (i.e. the neural network has *learned* the appropriate mapping) [1, Chap. 5].

Neural networks, in this general form, exhibit inherent scalability issues. More precisely, the fact that the number of *learnable parameters* depends on the data dimension leads to statistical, optimization and computational issues [2, Sec. 9.2]. Convolutional neural networks (CNNs) overcome these issues by regularizing the linear transform to exploit the inherent data structure. Specifically, CNNs replace the linear operation by a convolution with a bank of small-support linear time-invariant filters. Convolutions are operations defined on data presenting a regular structure (such as time series or images) that can be computed efficiently. Moreover, by forcing the learnable convolutional filters to be small, the number of learnable parameters becomes independent of the size of the data. This allows CNNs to scale and show remarkable performance in various classification and regression tasks involving images and time series (regular data) [3].

Regularizing the linear operation in neural networks to exploit the inherent structure of data was the fundamental insight that led CNNs to become the information processing tool of choice when dealing with images and time series [3]. Data stemming from network systems, however, rarely (if ever) exhibit such a regular structure [4]. This implies that regular convolutional operations no longer appropriately exploit the structure of the data, and as such, cease to be useful in processing network data. Several attempts have been made to regularize neural networks to exploit the topology underlying network data, giving rise to *graph neural networks* (GNNs). First, the use of an operator in the graph spectrum was proposed [5], and later an approximation in terms of Chebyshev polynomials was introduced [6], [7]. Several other models for regularizing the linear operation have also been proposed [8]–[10]. Additionally, non-convolutional graph neural networks have also emerged [11]–[13].

In this paper, we define graph convolutions (Sec. II) in analogy to time convolutions and give insight into how they exploit the underlying graph topology of network data, allowing for a decentralized computation. We then use graph convolutions to regularize the linear transform of neural networks giving rise to two different GNN architectures. Selection GNNs (Sec. III) view convolution as a diffusion operator. They offer a framework that encompasses other GNN architectures [5]–[9] as particular cases. Aggregation GNNs (Sec. IV) consider the weighted aggregation of neighboring values at each node. Then, they exploit the regular structure to process the aggregated sequence by means of a CNN. The movie recommendation problem is briefly overviewed to illustrate the effect of these two types of GNNs (Sec. V).

II. Graph Convolutions

Consider a datum $\mathbf{x} \in \mathcal{X}$ belonging to some field $\mathcal{X}$, and a target representation $\mathbf{y} \in \mathcal{Y}$, defined over some other field

Supported by NSF CCF 1717120, ARO W911NF1710438, ARL DCIST CRA W911NF-17-2-0181, ISTC-WAS and Intel DevCloud; and Spain MINECO grants No TEC2013-41604-R and TEC2016-75361-R.

$\mathcal{Y}$. Neural networks are used to parameterize the mapping between $\mathbf{x}$ and $\mathbf{y}$, and consist of a cascade of L layers, where each layer ℓ is a concatenation of a linear operation $\mathbf{A}_\ell$ followed by an activation function σ_ℓ (typically a pointwise nonlinearity)

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

for $\ell = 1, \ldots, L$, with $\mathbf{x}_0 = \mathbf{x}$. The output of the last layer is typically used as an estimate of the target representation $\hat{\mathbf{y}}(\mathbf{x}) = \mathbf{x}_L$. Given a training set $\mathcal{T} = \{(\mathbf{x}_{(m)}, \mathbf{y}_{(m)})\}_{m=1}^{|\mathcal{T}|}$, we determine the set of linear transforms $\{\mathbf{A}_\ell\}$ that *fit* the training set by minimizing some cost function over it $\sum_{\mathcal{T}} J[\mathbf{y}_{(m)}, \hat{\mathbf{y}}(\mathbf{x}_{(m)})]$. Successful *learning* occurs when the *trained* model (1) outputs accurate representations $\hat{\mathbf{y}}(\mathbf{x})$ on *unseen* data $\mathbf{x} \notin \mathcal{T}$.

The linear transform $\mathbf{A}_1$, which contains the learnable parameters of the first layer, depends on the size of the data $\mathbf{x}$. This implies that neural networks (1) do not scale to large-dimensional data due to statistical, optimization and computational costs, such as the curse of dimensionality or the need for an extremely large dataset. Convolutional neural networks (CNNs) address this issue by regularizing the linear transform, forcing $\mathbf{A}_\ell$ to be a convolution with a bank of small-support filters. The learnable parameters are now the filter taps, whose number can be set independently of the size of the data.

Equally important, the use of convolutions provides a linear operation that exploits the (regular) structure of data since it relates data elements that are spatially (images) or temporally (time signals) nearby, irrespective of their absolute location. For instance, let $\mathbf{x} = [x_1, \ldots, x_N] \in \mathbb{R}^N$ be a discrete-time signal and let $\mathbf{h} = [h_0, \ldots, h_{K-1}] \in \mathbb{R}^K$ be a set of K filter taps. Then, the output at time instant n of the convolution is

$$[\mathbf{x} * \mathbf{h}]_n = \sum_{k=0}^{K} h_k x_{n-k} \tag{2}$$

which is a linear combination of K consecutive time instants. The use of linear operations that exploit the underlying structure of the data to regularize neural networks is the key aspect that contributed to the remarkable success of CNNs.

In this paper, we are interested in network data, and therefore we want to determine a linear operation that takes into account the underlying network topology that describes this type of data. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ be a graph with a set of N nodes $\mathcal{V} = \{1, \ldots, N\}$, a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ and an edge weighing function $\mathcal{W} : \mathcal{E} \to \mathbb{R}$. Network data $\mathbf{x}$ is described by associating a vector to each of the nodes of this graph $\mathbf{x} : \mathcal{V} \to \mathbb{R}^F$. We denote $\mathbf{x} = \{\mathbf{x}^f\}_{f=1}^N$ as a collection of F *graph signals* $\mathbf{x}^f \in \mathbb{R}^N$, where each element $[\mathbf{x}^f]_i = x_i^f$ represents the value of *feature* f at node i.

Graph $\mathcal{G}$ determines the pairwise relationship between data elements imposed by the network. To make this dependence more explicit, we associate a *graph shift operator* (GSO) $\mathbf{S}$ to graph $\mathcal{G}$. A GSO is a matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$ that respects the sparsity of the graph, i.e., $[\mathbf{S}]_{ij} = s_{ij}$ can be nonzero if and only if $(j, i) \in \mathcal{E}$ or $i = j$. Examples of GSOs found in the literature [4] include the adjacency matrix, the graph Laplacian, and several normalized versions of these graph operators [6], [7]. The GSO can then be used to define a linear operation $\mathbf{S}\mathbf{x}^f$ whose output, due to the sparsity of $\mathbf{S}$, is computed by means of local exchanges only,

$$[\mathbf{S}\mathbf{x}^f]_i = \sum_{j \in \mathcal{N}_i} s_{ij} x_j^f \tag{3}$$

where $\mathcal{N}_i$ is the set of neighboring nodes of node i. This operation *shifts* the value of the signal around the graph to its one-hop neighbors, that perform a weighted sum following the weights in the GSO $\mathbf{S}$. This operation can be applied repeatedly $\mathbf{S}^k\mathbf{x} = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x})$ and implemented locally by k repeated exchanges of information with the one-hop neighbors. The output of $\mathbf{S}^k\mathbf{x}$ gathers, at each node, a summary of the information located up to the k-hop neighborhood.

In analogy to time convolutions (2), we define a *graph convolution* as a linear combination of shifted versions of the signal

$$\mathbf{h} *_{\mathbf{S}} \mathbf{x}^f = \sum_{k=0}^{K-1} h_k \, \mathbf{S}^k \mathbf{x}^f \tag{4}$$

where $\mathbf{h} = [h_0, \ldots, h_{K-1}] \in \mathbb{R}^K$ is the set of K filter taps. The graph convolution (4) thus computes a linear combination of the values contained in consecutive neighborhoods up to the K-hop one. We note that (4) boils down to (2) when $\mathbf{S}$ represents the adjacency matrix of a directed-cycle (the support of discrete-time signals).

Graph convolutions (4) can be analyzed in terms of the spectrum of the GSO, also known as the *graph frequency domain* [4]. Let the GSO admit an eigendecomposition in terms of an orthonormal basis of eigenvectors $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathsf{H}}$. The *graph Fourier transform* (GFT) of a graph signal is then defined as the projection of the signal onto the eigenvector basis of the GSO, $\tilde{\mathbf{x}}^f = \mathbf{V}^{\mathsf{H}}\mathbf{x}^f$. Noting that the output of the graph convolution (4) is another graph signal $\mathbf{y}^f = \mathbf{h} *_{\mathbf{S}} \mathbf{x}^f$ we can compute the GFT of the output as

$$\tilde{\mathbf{y}}^f = \mathbf{V}^{\mathsf{H}}\mathbf{y}^f = \sum_{k=0}^{K-1} h_k \mathbf{\Lambda}^k \tilde{\mathbf{x}}^f = \operatorname{diag}(\tilde{\mathbf{h}})\tilde{\mathbf{x}}^f = \tilde{\mathbf{h}} \circ \tilde{\mathbf{x}}^f \tag{5}$$

where $\circ$ denotes elementwise multiplication (Hadamard product). We observe that graph convolutions (4) can be computed as multiplications in the graph frequency domain (5), in analogy to the convolution theorem. Note that the GFT of the filter taps $\tilde{\mathbf{h}}$ is computed by means of a polynomial function of the eigenvalues $[\tilde{\mathbf{h}}]_i = \sum_{k=0}^{K-1} h_k \lambda_i^k$, whereas the GFT of a graph signal depends on the eigenvectors of the GSO.

III. Selection GNNs

Graph convolutions defined as in (4) are linear operations that exploit the structure of data, so we use them to regularize the linear transform in neural networks (1), to obtain a *graph neural network* (GNN)

$$\mathbf{x}_\ell^f = \sigma_\ell\left(\sum_{g=1}^{F_{\ell-1}} \mathbf{h}_\ell^{fg} *_{\mathbf{S}} \mathbf{x}_{\ell-1}^g\right) = \sigma_\ell\left(\sum_{g=1}^{F_{\ell-1}} \sum_{k=0}^{K_\ell - 1} h_{\ell k}^{fg} \mathbf{S}^k \mathbf{x}_{\ell-1}^g\right) \tag{6}$$

for $\ell = 1, \ldots, L$, with $\mathbf{x}_0^f = \mathbf{x}^f$ as input. For each layer, there is a set of filter banks $\{\mathbf{h}_\ell^{fg}\}$, $f = 1, \ldots, F_\ell$,

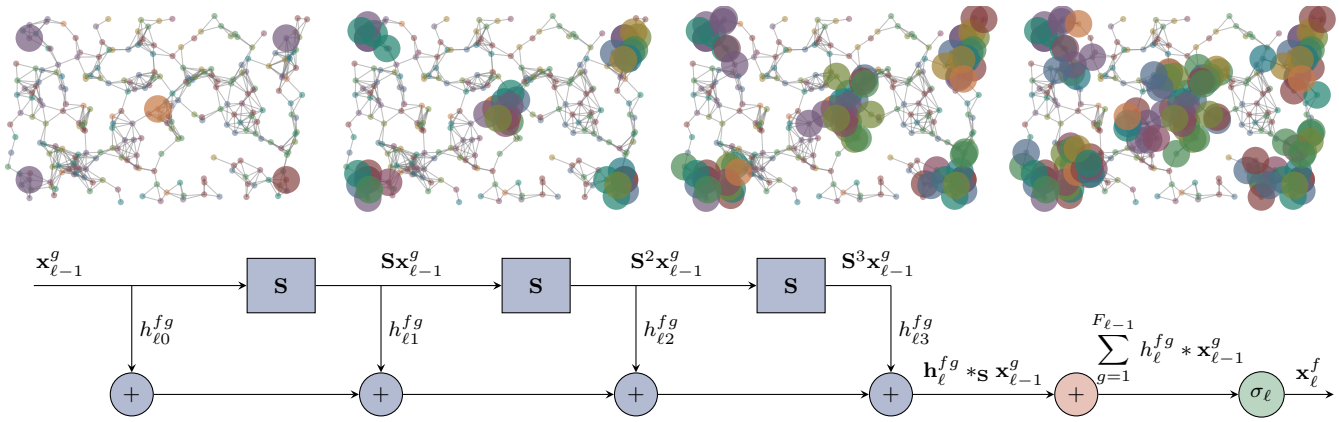


Figure 1: Selection GNNs (6). Every node takes its data value $\mathbf{x}^g_{\ell-1}$ and weighs it by $h^{fg}_{\ell 0}$ (first graph). Then, all the nodes exchange information with their one-hop neighbors to build $\mathbf{S}\mathbf{x}^g_{\ell-1}$, and weigh the result by $h^{fg}_{\ell 1}$ (second graph). Next, they exchange their values of $\mathbf{S}\mathbf{x}^g_{\ell-1}$ again to build $\mathbf{S}^2\mathbf{x}^g_{\ell-1}$ and weigh it by $h^{fg}_{\ell 2}$ (third graph). This procedure continues for K steps until all $h^{fg}_{\ell k}\mathbf{S}^k\mathbf{x}^g_{\ell-1}$ have been computed for $k = 0, \ldots, K-1$, and added up to obtain the output of the graph convolution operation (4), namely $\mathbf{h}^{fg}_{\ell} *_{\mathbf{S}} \mathbf{x}^g_{\ell-1}$. Then this output is added up with the outputs of the other $F_{\ell-1}$ filters in the bank (red sum; not shown to avoid cluttering) and the nonlinearity σ_ℓ applied to compute $\mathbf{x}^f_{\ell}$. To avoid cluttering, this operation is illustrated on only 5 nodes. In each case, the corresponding neighbors accessed by successive relays of information are indicated by the colored disks.

$g = 1, \ldots, F_{\ell-1}$ relating features at different layers, which contain the $K_\ell F_\ell F_{\ell-1}$ learnable parameters $\{h^{fg}_{\ell k}\}$, a number independent of the size of the graph N. This straightforward implementation of the graph convolution (4) to regularize the linear transform in (1) is termed *selection GNN*, and is depicted in Fig. 1.

Selection GNNs (6) can be implemented efficiently in a decentralized fashion since the graph convolutions are local and the nonlinearities are pointwise. The effect of implementing the GNN as in (6) is to diffuse, or percolate, the signal through the graph. The graph convolution then amounts to rescaling each diffusion step (containing information from the k-hop neighborhood) by a different weight. Every node needs to collect the value of the diffusion at each step, weigh it by the corresponding filter tap, and then add it up together. We observe that, since the graph convolution (4) boils down to a time convolution (2) on a directed cycle graph (the support of discrete-time signals), then the selection GNN (6) boils down to a traditional CNN on this support.

Architecture (6) serves as a general model for many of the existing GNNs in the literature [5]–[9]. We note that [5] uses the frequency representation (5) of graph convolutions to design the linear operations (we note that any analytic function $\tilde{\mathbf{h}}$ of the eigenvalues on a finite graph can be written as a polynomial with at most $K = N$ coefficients, thus leading to (6)). The model in [8] sets $F_1 = NF_0$ in (6) and uses filters with a single coefficient $[\mathbf{h}^{fg}_1]_k$, so that each filter in the bank gathers information from exactly the k-hop neighborhood. In [6], [7], GNNs (6) are considered only for specific choices of normalized GSOs, and consist of Chebyshev polynomials which are particular computational implementations of (4). Finally, [9] considers GNN (6) that gathers information up to the one-hop neighborhood $K = 2$ and also shares the weights between the two coefficients $[\mathbf{h}^{fg}_\ell]_0 = [\mathbf{h}^{fg}_\ell]_1$. Other weight-sharing schemes can be found in [13], [14]. The generalization potential of graph convolutions as in (4) was noted in [10].

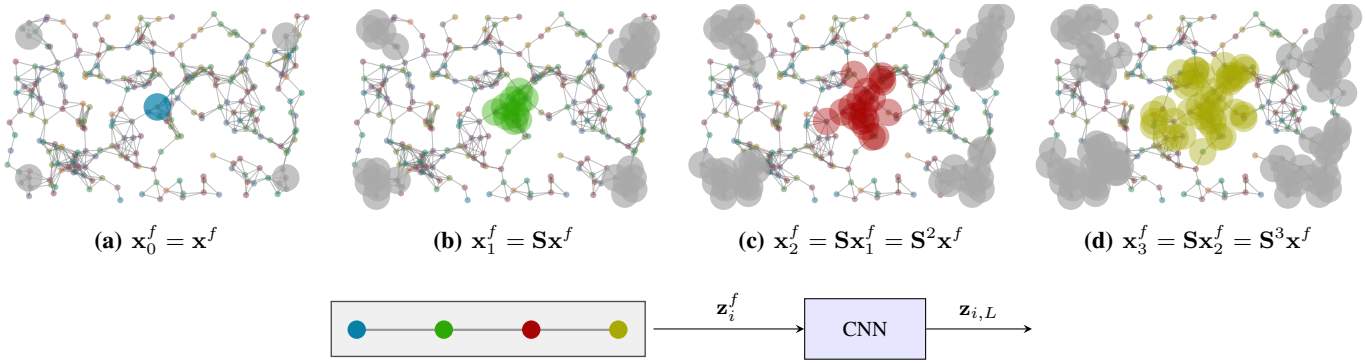
Remark 1 (Pooling)**.** When operating on images, CNNs usually add a pooling operation to keep the dimension of the data at each layer under control. Pooling basically consists in reducing the size of the image by computing summaries (typically, max- or average-) of increasingly bigger regions of the image. In this way, at each layer, the image to be processed is smaller. Therefore, even though more features are computed at each layer, the total number of data elements (the product of the number of features and the size of the images) is overall decreasing (trading off spatial information -image size- for feature information). When dealing with network data, pooling does not seem to be so crucial since the operations are carried out independently at each node, naturally offloading the computational cost (which is not the case for images, where pixels represent units of information, but do not have computational capabilities). Nonetheless, pooling operations for GNNs have been developed, including graph coarsening [6] and zero-padding [15].

IV. Aggregation GNNs

Graph convolution (4) also allows for a different interpretation. Instead of understanding it as a *weighted diffusion* of the signal values through the graph $(h_k\mathbf{S}^k)\mathbf{x}^f$, we can think of it as a *weighted aggregation* of values at a given node $h_k(\mathbf{S}^k\mathbf{x}^f)$. More specifically, consider some node $i \in \mathcal{V}$, and build the sequence of aggregated values at that node

$$\mathbf{z}^f_i = \mathbf{z}_i(\mathbf{x}^f) = \Big[[\mathbf{x}^f]_i, [\mathbf{S}\mathbf{x}^f]_i, [\mathbf{S}^2\mathbf{x}^f]_i, \ldots, [\mathbf{S}^{K-1}\mathbf{x}^f]_i\Big]. \quad (7)$$

Then, the output of the graph convolution (4) at that node, can be computed as $[\mathbf{h} *_{\mathbf{S}} \mathbf{x}^f]_i = \mathbf{h}^{\mathsf{T}}\mathbf{z}^f_i$. Likewise, the output of the GNN (6) at node i can be computed as $[\mathbf{x}^f_\ell]_i = \sigma_\ell(\sum_{g=1}^{G}(\mathbf{h}^{fg}_\ell)^{\mathsf{T}}\mathbf{z}_i(\mathbf{x}^g_{\ell-1}))$. So, essentially, we have just reformulated the graph convolution, focusing on the node



(a) $\mathbf{x}_0^f = \mathbf{x}^f$ **(b)** $\mathbf{x}_1^f = \mathbf{S}\mathbf{x}^f$ **(c)** $\mathbf{x}_2^f = \mathbf{S}\mathbf{x}_1^f = \mathbf{S}^2\mathbf{x}^f$ **(d)** $\mathbf{x}_3^f = \mathbf{S}\mathbf{x}_2^f = \mathbf{S}^3\mathbf{x}^f$

(e) Processing at the i-th node

Figure 2: Aggregation GNN (9). For a graph with GSO $\mathbf{S}$, perform successive local exchanges between nodes and their neighbors. For each k-hop neighborhood, record local components of the signal $\mathbf{x}_k^f = \mathbf{S}^k\mathbf{x}^f$. This process builds a sequence of signals $\mathbf{z}_i^f$ that aggregate the information of neighborhoods of subsequent depth, cf. (7). The signals accumulated at each node are such that they can be processed with a local convolutional neural network (CNN) to produce output $\mathbf{z}_{i,L}$.

collecting all the relevant information from its neighbors, and then linearly combining it.

The vector $\mathbf{z}_i^f$ (7) collected at each node, however, offers a key insight into the structure of this sequence of aggregated values: it has a regular time-like structure. Vector $\mathbf{z}_i^f$ exhibits a regular structure in the sense that consecutive elements in the vector represent values collected from immediate neighborhoods. In this way, $\mathbf{z}_i(\mathbf{x}^f)$ has transformed the graph signal $\mathbf{x}^f$ into a regular-structure signal, while still taking into account the underlying, irregular graph support. Now, if we have a signal $\mathbf{z}_i^f$ with regular structure, then we can apply regular convolutions (2)

$$[\mathbf{h} * \mathbf{z}_i^f]_n = \sum_{k=0}^{K-1} h_k [\mathbf{z}_i^f]_{n-k} = \sum_{k=0}^{K-1} h_k [\mathbf{S}^{n-k-1}\mathbf{x}^f]_i \qquad (8)$$

which, we see, amounts to a linear combination of neighboring values of the signal, akin to (4). If a regular convolution can be successfully applied to the aggregated sequence $\mathbf{z}_i^f$, then a CNN can be used on $\mathbf{z}_i^f$ as well. We thus define the *aggregation GNN* architecture as collecting the aggregated sequences $\mathbf{z}_i^f$ at every node i and processing them by a CNN,

$$\mathbf{z}_{i,\ell}^f = \sigma_\ell \left(\sum_{g=1}^{F_{\ell-1}} \mathbf{h}_\ell^{fg} * \mathbf{z}_{i,\ell-1}^g \right) \qquad (9)$$

for $\ell = 1, \ldots, L$, where we set $\mathbf{z}_{i,0}^f = \mathbf{z}_i(\mathbf{x}^f)$. The output of the aggregation GNN $\hat{\mathbf{y}}(\mathbf{x})$ is collected at all nodes $\hat{\mathbf{y}}(\mathbf{x}) = \{\mathbf{z}_{i,L}^f\}_{f,i}$. See Fig. 2 for an illustration of aggregation GNNs.

Aggregation GNNs exchange information with their neighbors K times, and then do the bulk of the processing internally, to output a feature $\mathbf{z}_{i,L}^f$. Therefore, aggregation GNNs rely heavily on the very efficient computation of regular convolutions, as well as in the computational power available at each node. Interestingly, processing on all nodes of the network might not be necessary. In one extreme case, we can select a single node $i \in \mathcal{V}$ and, if we set $K = N$, then this single node gathers all the information available in the graph signal, since, under some mild conditions, there is a one-to-one mapping between $\mathbf{z}_i^f$ and $\mathbf{x}^f$ [16]. In other words, if we are allowed to do $K = N$ exchanges of information, then a single node is capable of collecting the exact same information than in the original graph signal. Thus, a single node can do all the processing and compute the output. In any case, we note that, if not all nodes are used, then the performance of the aggregation GNN is contingent on the selected nodes.

V. Numerical Experiments

To illustrate the performance of both selection and aggregation GNNs, we consider the problem of movie recommendation [17], [18]. We use the MovieLens-100k dataset [19], which has a list of $100,000$ ratings given by 943 users to some of the $1,582$ movies available. The problem we are interested in is that of estimating the rating a user might give to some specific movie that they have not watched yet.

We consider a movie-based graph to support the data. Each movie is a node in the graph and we compute a similarity score between movies by taking into account the correlation among ratings given to each pair of movies by the same set of users, see [18, eq. (6)] for details. We use this score as edge weights, and we build a graph consisting of the 40-nearest neighbors. To build the training and test sets, we consider all users that have rated the specific movie we are interested in, and we split them 90% for training and 10% for testing[1]. Then, each user represents a graph signal, where the value at each node is the rating given to that movie. Movies not rated are given a value of 0. The value of the rating of the specific movie we are interested in is extracted as a label, and zeroed out in the graph signal. The objective is to use the ratings given by the user to other movies (nonzero elements in the graph signal) to estimate the rating they would give to the specific movie of interest.

The map between the graph signal (ratings for some of the movies) and the target (rating for the specific movie) is parameterized by a GNN. In particular, we consider 1- and 2-layer selection GNNs and 1- and 2-layer aggregation GNNs.

[1]No samples from the test set were used to build the graph support.

Architecture	RMSE
Selection 1 layer	$1.0486(\pm 0.1079)$
Selection 2 layers	$0.9870(\pm 0.1536)$
Aggregation 1 layer	$0.9274(\pm 0.1561)$
Aggregation 2 layers	$0.9274(\pm 0.1561)$
Graph Filter [18]	$1.1388(\pm 0.1600)$

Table I: Root mean squared error (RMSE) for predicting the rating given to the movie "Toy Story". Computed over the test set. Averaged over 10 different data splits.

We note that the aggregation GNN is computed at a single node $i \in \mathcal{V}$ with the highest experimentally designed sampling (EDS) coefficient [20] taking into account $K = 10$ neighbor exchanges. In all cases, we set $F_1 = 64$ and $F_2 = 32$ and use filters of $K_1 = K_2 = 5$ taps. The chosen activation function is a ReLU. We follow with a fully-connected layer mapping the output features of the GNN to a 5-dimensional vector, followed by a sigmoid, to represent the probability of each of the 5 possible ratings (integer values between 1 and 5). We benchmark the prediction performance with that of linear graph filters, whose application to the problem of recommender systems was proposed in [18], where the obtained performance is compared with that of traditional collaborative filtering methods.

We train these models using an ADAM optimizer [21] with learning rate 0.005, and decaying factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We train for 40 epochs with batch sizes of 5 samples. We evaluate the models computing the RMSE on the test set, to take into account the difference in ratings (i.e. how different our predicted rating is from the true one is important). Results when the movie of interest is "Toy Story"[2] are shown in Table I. The first observation is that the best performing architecture is the Aggregation GNN (19% improvement over graph filters). We also observe that there is no gain from including one extra layer. We then observe that the selection GNN also improves the performance over the graph filter (8% for 1-layer, 13% for the 2-layer), which accounts for the importance of the nonlinearity.

VI. Conclusions

We addressed the problem of regularizing neural networks to make them suitable for processing network data. We introduced graph convolutions by analogy to time convolutions (linear combination of shifted versions of the signal) and used it as a proper regularization that takes into account the underlying network structure of the data. We introduced two different GNN architectures stemming from different interpretations of the graph convolution. Namely, selection GNNs by understanding the graph convolution as a diffusion process, and aggregation GNNs by implementing the graph convolution as an aggregation sequence on the node. We illustrated the proposed architectures in the movie recommendation problem.

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. New York, NY: Springer, 2006.

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

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

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

[5] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *Int. Conf. Learning Representations 2014*. Banff, AB: Assoc. Comput. Linguistics, 14-16 Apr. 2014, pp. 1–14.

[6] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Annu. Conf. Neural Inform. Process. Syst. 2016*. Barcelona, Spain: NIPS Foundation, 5-10 Dec. 2016, pp. 3844–3858.

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Int. Conf. Learning Representations 2017*. Toulon, France: Assoc. Comput. Linguistics, 24-26 Apr. 2017, pp. 1–14.

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

[9] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Int. Conf. Learning Representations 2019*. New Orleans, LA: Assoc. Comput. Linguistics, 6-9 May 2019.

[10] J. Du, J. Shi, S. Kar, and J. M. F. Moura, "On graph convolution for graph cnns," in *2018 IEEE Data Sci. Workshop*. Lausanne, Switzerland: IEEE, 4-6 June 2018, pp. 239–243.

[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Int. Conf. Learning Representations 2018*. Vancouver, BC: Assoc. Comput. Linguistics, 30 Apr.-3 May 2018, pp. 1–12.

[12] 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, pp. 220–224.

[13] E. Isufi, F. Gama, and A. Ribeiro, "Generalizing graph convolutional neural networks with edge-variant recursions on graphs," *arXiv:1903.01298v1 [cs.LG]*, 4 March 2019. [Online]. Available: http://arxiv.org/abs/1903.01298

[14] F. Gama, A. G. Marques, A. Ribeiro, and G. Leus, "MIMO graph filters for convolutional networks," in *19th IEEE Int. Workshop Signal Process. Advances in Wireless Commun.* Kalamata, Greece: IEEE, 25-28 June 2018, pp. 1–5.

[15] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 4, pp. 1034–1049, Feb. 2019.

[16] 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.

[17] L. Ruiz, F. Gama, A. G. Marques, and A. Ribeiro, "Invariance-preserving localized activation functions for graph neural networks," *arXiv:1903.12575v1 [eess.SP]*, 29 March 2019. [Online]. Available: http://arxiv.org/abs/1903.12575

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

[19] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," *ACM Trans. Interactive Intell. Syst.*, vol. 5, no. 4, pp. 19:(1–19), Jan. 2016.

[20] 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.

[21] D. P. Kingma and J. L. Ba, "ADAM: A method for stochastic optimization," in *Int. Conf. Learning Representations 2015*. San Diego, CA: Assoc. Comput. Linguistics, 7-9 May 2015, pp. 1–15.

[2]This movie was selected because it was rated 452 times, the largest number of ratings available, making possible to build larger datasets.