Overparametrized Deep Encoder-Decoder Schemes for Inputs and Outputs Defined over Graphs

Samuel Rey, Victor Tenorio, Sergio Rozada, Luca Martino, and Antonio G. Marques*

***Abstract*—There is a growing interest in the joint application of graph signal processing and neural networks (NNs) for learning problems involving complex, non-linear and/or non-Euclidean datasets. This paper proposes an overparametrized graph-aware NN architecture able to represent a non-linear mapping between two graph signals, each defined on a different graph. The considered architecture is based on two NNs and a common latent space. Specifically, we consider an overparametrized graph-aware NN encoder which maps the input graph signal to a latent space, followed by an overparametrized graph-aware NN decoder that transforms the latent representation to the output graph signal. The parameters of the two NNs are jointly tuned by applying the back-propagation algorithm with an early stopping procedure to prevent overfitting. The overall architecture can be interpreted as an overparametrized graph-aware encoder/decoder NN operating over two different graphs. A key element in the encoder (decoding) scheme is the consideration of a nested collection of parametric graph-aware (down-) up-sampling operators, whose design will be studied in detail. We show by numerical simulations that the proposed scheme outperforms the corresponding benchmark NN architectures, previously introduced in the literature.**

***Index Terms*—Graph Neural Networks, Graph Autoencoders, Non-Euclidean Data, Geometric Deep Learning.**

I. Introduction

Scientific and societal progress requires processing, understanding and learning from vast amounts of complex data, calling for new models and tools able to account for irregular information domains and nonlinear interactions.

Relevant examples of signals defined on graphs, which are a versatile tool to model irregular domains, range from patterns of neurological activity defined on brain networks to social networks where epidemics spread [1]. Motivated by this, the graph signal processing (GSP) community has recently studied how classical signal processing (SP) concepts can be defined over graphs. Early efforts investigated linear graph filters, graph Fourier representations or inverse problems. [2]–[5]. Recent trends have shifted the interest towards higher-dimensional graph signals and nonlinear GSP architectures, including median filters [6] and deep learning (DL) architectures [7], with a special interest in convolutional schemes [8], [9]. Graph-aware DL schemes are indeed receiving considerable attention [10]–[15]. This paper proposes to take advantage of recent successes in GSP and DL, leveraging a graph-aware overparametrized encoder-decoder architecture that outperforms a benchmark of NN models.

This work was supported by the Spanish Federal Grants Klinilycs (TEC2016-75361-R), SPGRAPH (PID2019-105032GB) and Instituto de Salud Carlos III DTS17/00158, FPU17/04520; and by the Young Researchers R&D Project Grants F661-MAPPING-UCI and F663-AAGNCS both funded by the Community of Madrid (CAM) and the King Juan Carlos University (URJC). All the authors are with the Dept. of Signal Theory and Comms., URJC, Madrid, Spain. *Contact email : antonio.garcia.marques (AT) urjc.es.

Our goal is to learn a mapping from a graph-signal input space to a graph-signal output space, where the graphs supporting both spaces are different. Multirelational graphs can be easily expressed in this way. In order to address this problem, we assume that there exists a common latent space where the signals can be represented. Then, we split the mapping into two steps. The first one maps the input space into the latent space and the second one maps the latent space into the output space. This approach resembles autoencoders, which first map the input into a reduced space and then try to reconstruct the input from there. Therefore, we propose using two nonlinear graph-aware NNs forming an encoder/decoder, where each NN depends on its particular graph (input or output) and has its own set of parameters.

The mapping from the signal space to the latent space is given by a graph-aware encoder. The encoder is a parametric architecture composed of a sequence of smaller (coarser) graphs generated by an agglomerative hierarchical clustering algorithm [16]. Every encoding layer will also consider a down-sampling operator, some learnable parameters to mix the different features, and a point-wise nonlinearity. The decoder architecture is obtained by reversing the order of the layers and running the clustering algorithm on the output graph [15]. The resulting architecture is a parametrized graph-signal encoder/decoder. Recent work shows that untrained and underparametrized non-convolutional graph-aware settings can outperform state-of-the-art NN architectures [17].

In this work, we present an *overparametrized* graph-signal encoder/decoder that improves previous results, as we will show in the simulations. In the context of encoding, underparametrization forces the model to compress as much information as possible in a concise set of weights. It also protects against overfitting, enhancing robustness against noise. However, if the set of parameters is too small, the hypothesis space may not contain a map that fits appropriately the input and the output graph-signals even in the training set, leading to underfitting. Alternatively, overparametrized settings can be used. Although the number of parameters is redundant and the efficiency is lost in the trade-off, the hypothesis space is surely rich enough to capture a well-fitted mapping. In overparametrized architectures, overfitting should be kept under control, so that the learned parameters perform well over both the training and the testing (unobserved) data. This trade-off between well-fitted models and the ability to generalize appropriately is known as the bias-variance trade-off. To avoid overfitting in overparametrized settings, we use early stopping during training: the validation error is measured on each epoch during training until it starts to raise. Then, the training is stopped, even if the number of epochs set initially is not reached.

II. Fundamentals of graph signal processing

This section presents basic concepts of GSP that will be used in the rest of the paper.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a directed graph, where $\mathcal{V}$ is the set of N nodes, and $\mathcal{E}$ is the set of edges, with $(i,j) \in \mathcal{E}$ if i is connected to (can influence) node j. The set $\mathcal{V}_i := \{\, j \,|(j,i) \in \mathcal{E}\}$ denotes the incoming neighborhood of node i. For a given $\mathcal{G}$, the (possibly non-symmetric) adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ has non-zero elements A_{ij} if and only if $(j,i) \in \mathcal{E}$. If $\mathcal{G}$ is unweighted, the elements A_{ij} are binary (0 if $(j,i) \notin \mathcal{E}$ and 1 if $(j,i) \in \mathcal{E}$). In this paper, we focus on the processing of graph signals (features) defined on $\mathcal{V}$. Such signals can be represented as a vector $\mathbf{x} = [x_1, \ldots, x_N]^T \in \mathbb{R}^N$ where the i-th entry represents the signal value at node i. Since the signal $\mathbf{x}$ is defined on the graph $\mathcal{G}$, the underlying assumption in GSP is that the properties of $\mathbf{x}$ depend on the topology of $\mathcal{G}$. For instance, let us consider that the graph encodes similarity. In this case, if the value of A_{ij} is high, then one would expect the signal values x_i and x_j to be similar or closely related.

The graph-shift operator (GSO). The GSO is defined as an $N \times N$ matrix $\mathbf{S}$ whose entry S_{ij} can be non-zero only if $i = j$ or $(j,i) \in \mathcal{E}$. Common choices for $\mathbf{S}$ are the adjacency matrix $\mathbf{A}$ or, if the graph is undirected, the symmetric graph Laplacian $\mathbf{L}$, which is defined as $\mathbf{L} := \text{diag}(\mathbf{A1}) - \mathbf{A}$ [1], [18] where $\mathbf{1}$ is the all-ones vector. The GSO accounts for the topology of the graph and, at the same time, it represents a linear transformation that can be computed *locally*. Specifically, if $\mathbf{y} = [y_1, \ldots, y_N]^T$ is defined as $\mathbf{y} = \mathbf{Sx}$, then node i can compute y_i provided that it has access to the values of x_j at its neighbors $j \in \mathcal{V}_i$.

Graph filtering. Another important element in GSP is graph filtering. Graph filters are linear graph-signal operators that can be expressed as a polynomial of the GSO, i.e.,

$$\mathbf{F}(\mathbf{f}, \mathbf{S}) = \sum_{k=0}^{K} f_k \mathbf{S}^k, \tag{1}$$

where $\mathbf{f} = [f_0, ..., f_K]^T$ is a vector of $K+1$ coefficients of the filters and $\mathbf{S}$ is the GSO. Note that the graph filter $\mathbf{F} = \mathbf{F}(\mathbf{f}, \mathbf{S})$ is an $N \times N$ matrix, where we have explicitly denoted the dependence on $\mathbf{f}$ and $\mathbf{S}$. When applied to an input graph signal $\mathbf{x}$, the output of the filter $\mathbf{F}$ is

$$\mathbf{y} = \mathbf{F}(\mathbf{f}, \mathbf{S})\mathbf{x} = \sum_{k=0}^{K} f_k (\mathbf{S}^k \mathbf{x}) = \sum_{k=0}^{K} f_k \widetilde{\mathbf{x}}_k,$$

where $\widetilde{\mathbf{x}}_k = \mathbf{S}^k \mathbf{x}$ can be viewed as a version of $\mathbf{x}$ diffused across a k-hop neighborhood and f_k are coefficients which linearly combine the $K+1$ diffused signals $\{\widetilde{\mathbf{x}}_k\}_{k=0}^{K}$ [4].

III. Problem statement and proposed scheme

In the proposed approach, we consider two classes of graph signals: the input vector signals $\mathbf{x} \in \mathbb{R}^N$, defined on the graph $\mathcal{G}_X$ with N nodes and with GSO $\mathbf{S}_X$, and the output vector signals $\mathbf{y} \in \mathbb{R}^M$, laying on the graph $\mathcal{G}_Y$ with M nodes and with GSO $\mathbf{S}_Y$. The main purpose of this paper is to infer the *nonlinear mapping* $\mathbf{\Psi} : \mathbb{R}^N \to \mathbb{R}^M$,

$$\mathbf{y} = \mathbf{\Psi}(\mathbf{x} \,|\mathcal{G}_X, \mathcal{G}_Y), \tag{2}$$

taking into account the structure of the graphs $\mathcal{G}_X$ and $\mathcal{G}_Y$, by learning an NN architecture. To achieve this, we consider the existence of a common underlying linear space of dimensions $N_Z \times H_Z$, and then learn the transformations from the two graph-signal spaces to this latent space. Namely, denoting by $\mathbf{Z} \in \mathbb{R}^{N_Z \times H_Z}$ the representation (coordinates) in the latent space, we obtain two nonlinear transformations $\mathbf{F}_{\mathbf{\Theta}_X} : \mathbb{R}^N \to \mathbb{R}^{N_Z \times H_Z}$ and $\mathbf{F}_{\mathbf{\Theta}_Y} : \mathbb{R}^M \to \mathbb{R}^{N_Z \times H_Z}$, such that

$$\mathbf{F}_{\mathbf{\Theta}_X}(\mathbf{x}|\mathcal{G}_X) = \mathbf{Z}, \tag{3}$$
$$\mathbf{F}_{\mathbf{\Theta}_Y}(\mathbf{y}|\mathcal{G}_Y) = \mathbf{Z}. \tag{4}$$

We also denote as $\mathbf{F}^{-1}_{\mathbf{\Theta}_Y} : \mathbb{R}^{N_Z \times H_Z} \to \mathbb{R}^M$ the inverse mapping from $\mathbf{Z}$ to $\mathbf{y}$. These transformations can be combined to define the desired complete mapping $\mathbf{\Psi}$ as

$$\mathbf{y} = \mathbf{\Psi}(\mathbf{x}|\mathcal{G}_X, \mathcal{G}_Y) = \mathbf{F}^{-1}_{\mathbf{\Theta}_Y}(\mathbf{F}_{\mathbf{\Theta}_X}(\mathbf{x}|\mathcal{G}_X)|\mathcal{G}_Y). \tag{5}$$

The overall proposed approach is represented in Fig. 1. Note that we are also obtaining implicitly the transformations from the two graph spaces to the common latent domain $\mathbf{Z}$, as given in (3)-(4). This can be of interest, revealing insights about the data. Moreover, the matrix $\mathbf{Z}$ may be seen as a signal defined over a coarsened graph common to $\mathcal{G}_X$ and $\mathcal{G}_Y$.

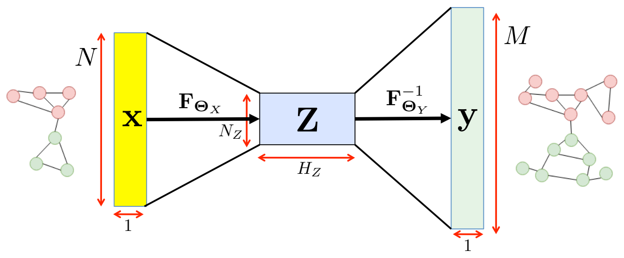

N
x
$\mathbf{F}_{\mathbf{\Theta}_X}$
N_Z
Z
H_Z
$\mathbf{F}^{-1}_{\mathbf{\Theta}_Y}$
y
M
1
1

Fig. 1. Graphical representation of the proposed architecture.

The main steps of the proposed scheme are described below:

1) We design an overparametrized graph NN scheme by selecting the number of layers for the encoder and the decoder and the parameters defining their sizes.
2) We decide in advance the size of the common latent space, i.e., the dimension of the $N_Z \times H_Z$ matrix $\mathbf{Z}$.
3) We define down-sampling and up-sampling operators to modify the size of the input and output graphs.

Please note that the information of the first graph $\mathcal{G}_X$ is used in (3), whereas the information about the second graph $\mathcal{G}_Y$ is employed in (4), and all the parameters contained in the matrices $\{\mathbf{\Theta}_X^{(\ell)}\}_{\ell=0}^{L_X}$, $\{\mathbf{\Theta}_Y^{(\ell)}\}_{\ell=0}^{L_Y}$ are simultaneously learned by using the back-propagation procedure [19].

IV. Overparametrized Graph-aware en/de-coder

In this section, we describe the design of the nonlinear mapping $\mathbf{\Psi}$ employing an overparametrized graph-aware encoder/decoder. The key element of the architecture is the design of a sequence of coarser graphs along with *up-sampling and down-sampling operators* that relate the values of the signals across the sequence of graphs. The imposed graph-aware structure helps the NNs to avoid "learning" the noise present in the data. We next describe the operation of the encoder, while the design of the down-sampling and up-sampling operators will be explained in detail in Section V.

Introduction to the graph-aware en/decoder architecture. Autoencoders are specific NN architectures with the particular characteristic that they map the input data into a reduced latent space, which can be interpreted as a compressed/summarized version of the original data. Similarly, the key piece of the proposed model is a non-linear transformation which brings an input graph signal $\mathbf{x}$ to the common matrix $\mathbf{Z}$. Therefore, graph-aware autoencoders emerge as a natural choice for the desired architecture. The idea of estimating a graph output signal from an input graph signal by going through a common latent reduced representation is related to Canonical Correlation Analysis (CCA), although in that case the mappings are linear and the inputs and outputs live in an Euclidean space.

Recall that $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{y} \in \mathbb{R}^M$ are the input and output signals, and $\mathbf{Z} \in \mathbb{R}^{N_Z \times H_Z}$ is the common matrix containing the hidden variables with $(N_Z H_Z) << N$, $(N_Z H_Z) << M$.

Encoder. We denote with L_X the number of layers of the encoder and with $\{N_X^{(l)}\}_{l=0}^{L_X}$ and $\{H_X^{(l)}\}_{l=0}^{L_X}$ the number nodes and features in each layer. Furthermore, the function $\mathbf{F}_{\mathbf{\Theta}_X} : \mathbb{R}^N \to \mathbb{R}^{N_Z \times H_Z}$ is a parametric nonlinear mapping whose parameters are collected into the matrices $\{\mathbf{\Theta}_X^{(\ell)}\}_{\ell=1}^{L_X}$, each one with dimension $H_X^{(\ell-1)} \times H_X^{(\ell)}$. The transformation $\mathbf{F}_{\mathbf{\Theta}_X}$ is defined by the following recursion:

1) Set $\mathbf{X}^{(0)} = \mathbf{x}$,
2) Set, for all $\ell = 1, .., L_X$,

$$\mathbf{R}^{(\ell)} = \mathbf{D}^{(\ell)} \mathbf{X}^{(\ell-1)} \mathbf{\Theta}_X^{(\ell)}, \qquad (6)$$

$$\mathbf{X}_{ij}^{(\ell)} = g^{(\ell)}\Big([\mathbf{R}^{(\ell)}]_{ij}\Big), \quad \text{for all} \quad i, j. \qquad (7)$$

3) Finally, set $\mathbf{Z} = \mathbf{X}^{(L_X)}$.

Note that $\mathbf{D}^{(\ell)}$ and $g^{(\ell)}$ denote the down-sampling matrix and the entry-wise non-linearity at ℓ-th layer. $\mathbf{X}^{(\ell-1)} \in \mathbb{R}^{N_X^{(\ell-1)} \times H_X^{(\ell-1)}}$ denotes the input to the ℓ-th layer, which is a collection of $H_X^{(\ell-1)}$ features corresponding to the signal defined in the graph $\mathcal{G}_X^{(\ell-1)}$. Whereas, the auxiliary matrix $\mathbf{R}^{(\ell)} \in \mathbb{R}^{N_X^{(\ell)} \times H_X^{(\ell)}}$ is a collection of $H_X^{(\ell)}$ features corresponding to a graph signal defined in $\mathcal{G}_X^{(\ell)}$. Thus, following the recursion, the (generated) matrix $\mathbf{X}^{(\ell)} \in \mathbb{R}^{N_X^{(\ell)} \times H_X^{(\ell)}}$ is the output of layer ℓ, which serves as the input for layer $\ell + 1$. It is important to observe that the matrices $\mathbf{X}^{(\ell-1)}$ to $\mathbf{R}^{(\ell)}$ are linked by a linear transformation given in (6). More specifically, $\mathbf{R}^{(\ell)}$ is obtained after multiplying $\mathbf{X}^{(\ell-1)}$ from the left by the down-sampling matrix $\mathbf{D}^{(\ell)}$ and from the right by $\mathbf{\Theta}_X^{(\ell)}$. The matrix $\mathbf{D}^{(\ell)}$ relates the signals at graph $\mathcal{G}_X^{(\ell-1)}$ with those at graph $\mathcal{G}_X^{(\ell)}$, and its design, which is carried out offline before the recursion is run, is described in Section V-B. The matrix $\mathbf{\Theta}_X^{(\ell)}$ is a matrix of coefficients that are learned by the back-propagation procedure and are used to combine linearly the different features. The matrix $\mathbf{X}^{(\ell)}$ is obtained by applying the scalar nonlinear operator $g^{(\ell)} : \mathbb{R} \to \mathbb{R}$ to each to the entries of $\mathbf{R}^{(\ell)}$. It is important to remark that the dimensions $N_X^{(\ell)}$ and $H_X^{(\ell)}$ for each of the layers are design parameters decided in advance by the user, jointly with the down-sampling matrices $\mathbf{D}^{(\ell)}$ associated with the graphs $\mathcal{G}_X^{(\ell-1)}$ and $\mathcal{G}_X^{(\ell)}$, with $\ell = 1, ..., L_X$.

Decoder. As for the encoder, let L_Y be the number of layers, and $\{N_Y^{(l)}\}_{l=0}^{L_Y}$ and $\{H_Y^{(l)}\}_{l=0}^{L_Y}$ the number of nodes and features in each layer. The decoder is represented by a parametric function $\mathbf{F}_{\mathbf{\Theta}_Y}^{-1} : \mathbb{R}^{N_Z \times H_Z} \to \mathbb{R}^M$ with parameters $\mathbf{\Theta}_Y$, and is defined through the following recursion:

1) Set $\mathbf{Y}^{(0)} = \mathbf{Z}$,
2) Set, for all $\ell = 1, ..., L$,

$$\mathbf{P}^{(\ell)} = \mathbf{U}^{(\ell)} \mathbf{Y}^{(\ell-1)} \mathbf{\Theta}_Y^{(\ell)}, \qquad (8)$$

$$\mathbf{Y}_{ij}^{(\ell)} = g^{(\ell)}\Big([\mathbf{P}^{(\ell)}]_{ij}\Big), \quad \text{for all} \quad i, j, \qquad (9)$$

3) Finally, set $\mathbf{Y} = \mathbf{Y}^{(L)}$.

Here, $\mathbf{P}^{(\ell)} \in \mathbb{R}^{N_Y^{(\ell)} \times H_Y^{(\ell)}}$ plays the same role as $\mathbf{R}^{(\ell)}$ in the encoder, so that is a collection of intermediate graph signals; $\mathbf{Y}^{(\ell-1)} \in \mathbb{R}^{N_Y^{(\ell-1)} \times H_Y^{(\ell-1)}}$ is the input of layer ℓ; $\mathbf{U}^{(\ell)} \in \mathbb{R}^{N_Y^{(\ell)} \times N_Y^{(\ell-1)}}$ is an up-sampling matrix linking the nodes of graph $\mathcal{G}_Y^{(\ell-1)}$ with the nodes of graph $\mathcal{G}_Y^{(\ell)}$; matrix $\mathbf{\Theta}_Y^{(\ell)} \in \mathbb{R}^{H_Y^{(\ell-1)} \times H_Y^{(\ell)}}$ contains the coefficients (tuned using back-propagation) that combine the different features.

Please notice that, if required, additional operations at each layer of the encoder and the decoder, such as batch normalization, can be included to re-scale and bias the output features. However, for simplicity, we have ignored such additional operations.

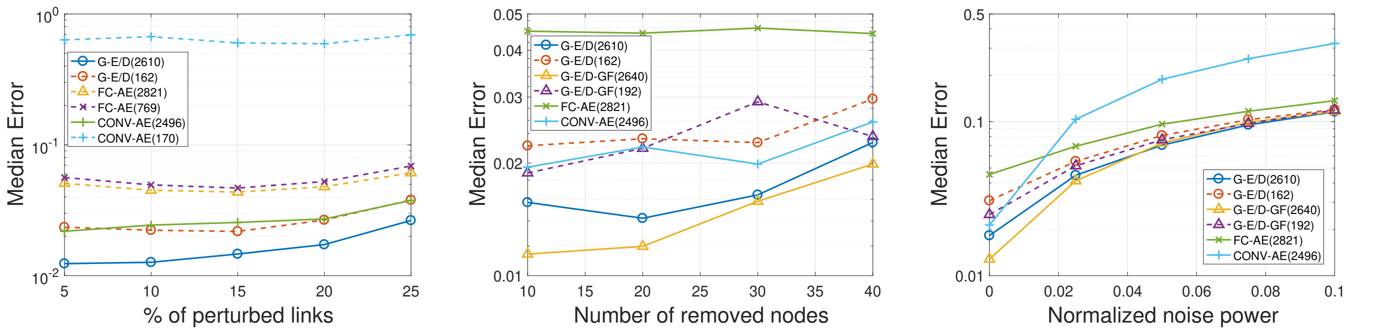
Remarks. Two main advantages of the proposed architecture are: (a) the domain information captured by the graphs is exploited by enforcing the values of the signal in neighboring nodes to be closely related, and (b) an efficient over-parametrization of the linear transformations in (6) and (8). As illustrated in the simulations, these features will render our design more effective and robust to noise perturbations.

V. Down-sampling and up-sampling operators

In the following section, we describe the construction of the down-sampling and up-sampling matrices $\mathbf{D}^{(\ell)}$ and $\mathbf{U}^{(\ell)}$, which is based on the clustering procedure given next.

A. *Hierarchical graph clustering*

Given $\mathcal{G}_X$ and $\mathcal{G}_Y$, our goal is to build down-sampling and up-sampling matrices that depend on the *topology* of the given graphs. The approach proposed in this paper relies



(a) (b) (c)

Fig. 2. Median error obtained on the test data after training the architectures listed in the legends (the numbers inside the parenthesis indicate the total number of parameters of the architecture). The median across 15 graphs is reported, with the remaining parameters being set as described in Section VI. Graph $\mathcal{G}_X$ is always created following an SBM with 256 nodes and 4 communities. Graph $\mathcal{G}_Y$ corresponds to different perturbed versions of $\mathcal{G}_X$ as described next: (a) Effect of removing links from $\mathcal{G}_X$; (b) Effect of removing nodes from $\mathcal{G}_X$; (c) Effect of removing 30 nodes and adding white noise to the test data.

on a clustering scheme that takes as input the graph of interest and returns as output a sequence of graphs, with a decreasing number of nodes. In our specific implementation, we have employed an agglomerative hierarchical clustering method that returns a dendrogram as output (rooted tree) [16]. The dendrogram can be interpreted as a nested collection of clusters, each one associated with a specific resolution (graph size). More specifically, we have L_X and L_Y different resolutions for the encoder and decoder, respectively. For the ℓ-th resolution, we obtain an $N_X^{(\ell)} \times N_X^{(\ell)}$ adjacency matrix $\mathbf{A}_X^{(\ell)}$ (and the $N_Y^{(\ell)} \times N_Y^{(\ell)}$ adjacency matrix $\mathbf{A}_Y^{(\ell)}$) associated with the clustered graph. Additionally, we define an $N_X^{(\ell)} \times N_X^{(\ell-1)}$ membership matrix $\mathbf{Q}_X^{(\ell)}$ (and $N_Y^{(\ell-1)} \times N_Y^{(\ell)}$ membership matrix $\mathbf{Q}_Y^{(\ell)}$) where the entry $[Q_X^{(\ell)}]_{ij}$ is non-zero only if node i in layer ℓ is the child of node j in layer $\ell-1$, as in [15].

B. Down-sampling and up-sampling operators

After applying the clustering, the resulting down-sampling and up-sampling operators are given by the matrices,

$$\begin{aligned} \mathbf{D}^{(\ell)} &= \mathbf{A}_X^{(\ell)}\mathbf{Q}_X^{(\ell)}, \\ \mathbf{U}^{(\ell)} &= \mathbf{A}_Y^{(\ell)}(\mathbf{Q}_Y^{(\ell)})^T, \end{aligned} \tag{10}$$

of dimensions $N_X^{(\ell)} \times N_X^{(\ell-1)}$ and $N_Y^{(\ell)} \times N_Y^{(\ell-1)}$, respectively. Note that, in general, the row of $\mathbf{Q}_X^{(\ell)}$ should be normalized to ensure that the magnitude of the signal does not depend on the number of children a parent has. For numerical and computational reasons (described in [15]), the following convex combinations

$$\begin{aligned} \mathbf{D}^{(\ell)} &= (\gamma\mathbf{I} + (1-\gamma)\mathbf{A}_X^{(\ell)})\mathbf{Q}_X^{(\ell)}, \\ \mathbf{U}^{(\ell)} &= (\gamma\mathbf{I} + (1-\gamma)\mathbf{A}_Y^{(\ell)})(\mathbf{Q}_Y^{(\ell)})^T. \end{aligned} \tag{11}$$

with $0 \leq \gamma \leq 1$, are often preferable. The γ parameter can be decided by the user in advance or, alternatively, it can be learned during the training process, jointly with the coefficients matrices $\{\mathbf{\Theta}_X^{(\ell)}\}_{\ell=1}^{L_X}$ and $\{\mathbf{\Theta}_Y^{(\ell)}\}_{\ell=1}^{L_Y}$.

C. Graph-filter based operators

The goal of this section is to leverage graph filters to design more general *parametric* and graph-aware down-sampling and up-sampling operators. To motivate this, note first that the matrix $\gamma\mathbf{I}+(1-\gamma)\mathbf{A}_X^{(\ell)}$ in (11) can be viewed as a graph filter of order $K=1$ with $\mathbf{S}_X = \mathbf{A}_X^{(\ell)}$, $f_0 = \gamma$ and $f_1 = 1-\gamma$. Hence, a natural next step is to consider graph filters of higher orders. Indeed, the use of low-pass filters as up-sampling operators has a long history in classical time-varying signal processing, with sincs being optimal interpolators for bandlimited signals. In this context, with $\mathbf{F}(\mathbf{f}_X^{(\ell)}, \mathbf{S}_X^{(\ell)})$ and $\mathbf{F}(\mathbf{f}_Y^{(\ell)}, \mathbf{S}_Y^{(\ell)})$ denoting graphs filters of order $K_X^{(\ell)}$ and $K_Y^{(\ell)}$, the following design for the down-sampling and up-sampling operators is proposed

$$\begin{aligned} \mathbf{D}_{GF}^{(\ell)} &= \mathbf{F}(\mathbf{f}_X^{(\ell)}, \mathbf{S}_X^{(\ell)})\mathbf{Q}_X^{(\ell)} = \left[\sum_{k=0}^{K_X^{(\ell)}} f_{k,X}^{(\ell)}(\mathbf{S}_X^{(\ell)})^k\right]\mathbf{Q}_X^{(\ell)}, \\ \mathbf{U}_{GF}^{(\ell)} &= \mathbf{F}(\mathbf{f}_Y^{(\ell)}, \mathbf{S}_Y^{(\ell)})(\mathbf{Q}_Y^{(\ell)})^T = \left[\sum_{k=0}^{K_Y^{(\ell)}} f_{k,Y}^{(\ell)}(\mathbf{S}_Y^{(\ell)})^k\right](\mathbf{Q}_Y^{(\ell)})^T. \end{aligned} \tag{12}$$

The vectors of coefficients $\mathbf{f}_X^{(\ell)}$, $\mathbf{f}_Y^{(\ell)}$ are learned together with the rest of the parameters. Also, the filter order may be set empirically or chosen as a prior. The graph filters endow the network with a greater level of expressiveness by allowing it to learn dependencies between nodes in a $K^{(\ell)}$-hop neighborhood. This enables the architecture to learn more complex mappings by only introducing a small number of additional parameters.

VI. Numerical Simulations

We use synthetic data for evaluating the proposed schemes and comparing them with other benchmark NN architectures. The input graphs $\mathcal{G}_X$ are generated from a Stochastic Block Model (SBM) [20] with 256 nodes, 4 communities and heterogeneous connection probabilities. The output graphs $\mathcal{G}_Y$ are created by random permutations and perturbations of $\mathcal{G}_X$ (additional details are given when introducing each of the test cases). The training pairs $\mathbf{x}$ and $\mathbf{y}$ correspond to diffused graph signals generated after applying the graph filter $\mathbf{F}_\mathbf{x}$ (alternatively $\mathbf{F}_\mathbf{y}$) to a random sparse vector $\mathbf{w}$ and implementing a median operator across the one-hop neighbors. For further implementation details, interested readers can check the code in GitHub[1]. The architectures are trained over 2000 synthetically-generated signals and tested over a set with 1000 signals. We compute the error as the median over 15 independent runs generating different graphs.

Test case 1: Graphs $\mathcal{G}_Y$ are created by randomly removing links from $\mathcal{G}_X$. Fig. 2(a) compares the performance of the Graph Encoder/Decoder architecture (G-E/D) with a fully-connected autoencoder (FC-AE) and a convolutional autoencoder (CONV-AE) [19], using an overparametrized and an underparametrized version of each of them. First of all, we observe that the error of our scheme decreases considerably as the number of parameters increases. The results also show that the underparametrized version of the graph-aware NN outperforms all non-graph-aware schemes, corroborating the benefits of incorporating the knowledge of the graphs into the processing scheme. Finally, we see that, when a higher percentage of links are removed, the error increases. This shows that as $\mathcal{G}_X$ and $\mathcal{G}_Y$ become more different, finding a good mapping between $\mathbf{x}$ and $\mathbf{y}$ becomes more challenging.

Test case 2: A different number of nodes are removed from $\mathcal{G}_X$ to create $\mathcal{G}_Y$. The results are depicted in Fig. 2(b). In addition, we test the performance of an enhanced graph-aware architecture, labelled as G-E/D-GF, where the down-sampling and up-sampling operators are based on learnable graph filters [cf. (12)]. Simulations reveal that, despite having only a few more training parameters, the new scheme clearly outperforms the tested alternatives, indicating that the graph filters are an efficient way to increase the expressiveness of the original G-E/D while leveraging the structure of the graph. Additionally, we also observe that, once again, the more different $\mathcal{G}_X$ and $\mathcal{G}_Y$ are, the more challenging the problem becomes.

Test case 3: In this setting, the graph $\mathcal{G}_Y$ is created by randomly removing 30 nodes from $\mathcal{G}_X$, as in the previous case. However, we add random noise, with a normalized power ranging from 0 to 0.1, to the testing set to evaluate the sensitivity of the different architectures. Results are reported in Fig. 2(c), showing how the graph-aware architectures outperform the rest of the schemes. In the absence of noise, the benefit of using graph filters is noticeable. However, as the noise power increases this effect vanishes and the performance of all the graph-aware architectures becomes more similar, including both the under and overparametrized schemes. Finally, it is worth noticing that the convolutional autoencoder is especially sensitive to the noise.

Due to time and space limitations, the development of alternative graph-aware architectures and its evaluation is left as future work.

[1]https://github.com/vmtenorio/GIGOArchitecture

VII. Conclusions

This work introduced an overparametrized deep graph-aware NN scheme to learn a transformation between graph signals $\{\mathbf{x}, \mathcal{G}_X\}$ to $\{\mathbf{y}, \mathcal{G}_Y\}$ defined on different graphs. The proposed architecture assumes the existence of a common latent linear space. The complete mapping is obtained by splitting the construction into two parts: one from the input graph domain to the latent space and another one from the latent space to the output graph domain. An overparametrized encoder (decoder) graph-aware NN architecture is proposed to learn that nonlinear transformation from (to) the input (output) graph domain to (from) the latent space. New down-sampling (up-sampling) operators based on graph filters are presented.

References

[1] 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.

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

[3] 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, Dic. 2015.

[4] 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.

[5] S. Segarra, G. Mateos, A. G. Marques, and A. Ribeiro, "Blind identification of graph filters," *IEEE Trans. Signal Process.*, vol. 65, no. 5, pp. 1146–1159, March 2017.

[6] S. Segarra, A. G. Marques, G. R. Arce, and A. Ribeiro, "Center-weighted median graph filters," in *Global Conf. Signal and Info. Process. (GlobalSIP)*, Dec 2016, pp. 336–340.

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

[8] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Process. Systems*, 2016, pp. 3844–3852.

[9] 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.

[10] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[11] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[13] L. Ruiz, F. Gama, and A. Ribeiro, "Gated graph convolutional recurrent neural networks," *arXiv preprint arXiv:1903.01888*, 2019.

[14] V. N. Ioannidis, A. G. Marques, and G. B. Giannakis, "A recurrent graph neural network for multi-relational data," in *IEEE Int. Conf. on Acoustics, Speech and Signal Process.*, 2019, pp. 8157–8161.

[15] S. Rey, A. G. Marques, and S. Segarra, "An underparametrized deep decoder architecture for graph signals," *IEEE Intl. Wrksp. Computat. Advances Multi-Sensor Adaptive Process. (CAMSAP)*, pp. 1–4, 2019.

[16] G. Carlsson, F. Memoli, A. Ribeiro, and S. Segarra, "Hierarchical clustering of asymmetric networks," *Advances in Data Analysis and Classification*, vol. 12, no. 1, pp. 65–105, Mar 2018.

[17] S. Rey, V. Tenorio, S. Rozada, L. Martino, and A. G. Marques, "Deep encoder-decoder neural network architectures for graph output signals," in *Asilomar Conf. Signals, Systems, and Computers*, 2019, pp. 1–5.

[18] P. M. Djurić and C. Richard (Eds.), *Cooperative and Graph Signal Processing*, Academic Press, 2018.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT Press, 2016.

[20] M. Newman, *Networks*, Oxford University Press, 2018.