GRAPH CONVOLUTIONAL NETWORKS WITH AUTOENCODER-BASED COMPRESSION AND MULTI-LAYER GRAPH LEARNING

Lorenzo Giusti[1], Claudio Battiloro[2], Paolo Di Lorenzo[2], Sergio Barbarossa[2]

[1]DIAG Department, Sapienza University of Rome, Via Ariosto 25, 00185, Rome, Italy
[2]DIET Department, Sapienza University of Rome, Via Eudossiana 18, 00184, Rome, Italy
E-mail: {lorenzo.giusti, claudio.battiloro, paolo.dilorenzo, sergio.barbarossa}@uniroma1.it

ABSTRACT

This work aims to propose a novel architecture and training strategy for graph convolutional networks (GCN). The proposed architecture, named Autoencoder-Aided GCN (AA-GCN), compresses the convolutional features in an information-rich embedding at multiple hidden layers, exploiting the presence of autoencoders before the pointwise nonlinearities. Then, we propose a novel end-to-end training procedure that learns different graph representations per layer, jointly with the GCN weights and auto-encoder parameters. As a result, the proposed strategy improves the computational scalability of the GCN, learning the best graph representations at each layer in a data-driven fashion. Several numerical results on synthetic and real data illustrate how our architecture and training procedure compares favorably with other state-of-the-art solutions, both in terms of robustness and learning performance.

***Index Terms*—** Deep learning, graph convolutional networks, graph signal processing, graph learning, autoencoder.

1. INTRODUCTION

Nowadays, graph-based data are pervasive, with applications ranging from social networks, recommender systems, cybersecurity, sensor networks, natural language processing, and many more [1]. In this context, GCNs represent the state-of-the-art solution for several machine learning tasks [2–4], where the flexibility of neural network models is coupled with prior knowledge about data relationships, which are expressed in terms of graph topology. Typical GCN models receive on input a multichannel signal and perform cascades of graph filtering operations (plus non-linearities) to extract meaningful features from the available data. Graph-filtering is typically performed employing convolution over graphs, which hinges on the interpretations of graph filters as polynomials in the graph shift operator [5–7]. However, two major challenges arise from the current architectures. First of all, it is often assumed that the topology of the initial graph, associated with the input data, remains constant throughout all the layers, thus implying that the relationships between the convolutional features extracted at all hidden layers do not change. Intuitively, this assumption becomes less and less valid as the depth of the GCN architecture increases. Furthermore, since the dimension of all the hidden layers convolutional features is typically kept constant and equal to the number of nodes of the initial graph, a computational scalability issue arises, as the number of data (i.e., the graph size) increases.

Related works. Some interesting solutions have been recently proposed to tackle the first challenge. In particular, the works in [8, 9] proposed an end-to-end graph learning framework for jointly and iteratively learning the GCN parameters and an optimal graph topology, as a refinement of the initially available graph. The work in [10] proposed a dynamic procedure for joint learning of graphs and GCN parameters based on pairwise similarities of convolutional features in each layer. In [11], the authors provided a method for joint learning of graph and GCN parameters based on solving a bilevel program that learns a discrete probability distribution on the edges of the graph. All these works focused on learning single or multi-layer graphs, jointly with the weights of the GCN, without however considering any compression mechanism to reduce the size of the convolutional features at the hidden layers. At the same time, several works have been proposed for pooling in GCNs, typically following a hierarchical scheme in which the pooling regions correspond to graph clusters that are combined to produce a coarser graph, see, e.g., [12–14]. Finally, the work in [3] proposed data reduction techniques for the hidden layers of GCNs, exploiting pooling and sampling methods from the graph signal processing (GSP) literature, although using the same initial graph in all the hidden layers. To the best of our knowledge, no works have considered the *joint* feature compression and multi-layer graph learning problem in GCNs.

Contributions. The goal of this paper is to propose a novel autoencoder-aided GCN architecture (AA-GCN), which compresses the convolutional features at the hidden layers while learning their most suitable per-layer graph representation. We exploit autoencoders in each layer, before applying the pointwise nonlinearity, so that the convolutional features can be tunably compressed in an information-rich embedding. Then, since compression calls for learning a new graph representation to be used in the following layer, we formulate a novel training strategy that *jointly* optimizes the GCN weights, the auto-encoder parameters, and the graph representations at different layers. Our strategy can work both for supervised and semi-supervised learning tasks, and the network is trained via usual backpropagation and projected-descent algorithms (e.g., gradient descent, ADAM, etc.). In this paper, for simplicity, we assume to have an initial graph that describes (or can be inferred from) the data, but the method can be straightforwardly extended whenever this prior knowledge is not available. Differently from [8, 9], we do not set the graph learning problem as a refinement of the initial graph to be used in all of the layers, but we enable the learning of a different graph per each layer. The work in [10] does not take into account pooling, sampling, or compression stages. Also, differently from [3], our architecture does not explicitly perform pooling and sampling stages, but we use autoencoders to compress the convolutional features, in a data-driven and goal-oriented fashion. The proposed strategy is then customized to two learning tasks, namely, authorship attribution, and source localization with noisy data. Numerical simulations illustrate the competitive performance of the proposed method, in terms of accuracy and robustness, when compared with other techniques available in the literature.

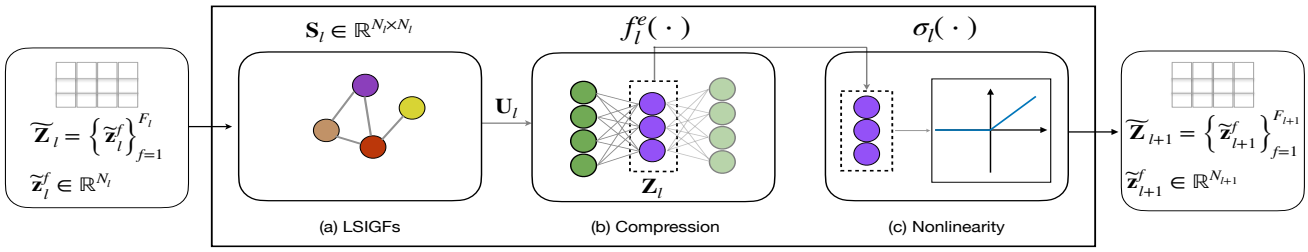

Fig. 1. AA-GCN Layer

2. BACKGROUND

In this section, we briefly recall the background useful to handle graph convolutional networks. Consider a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consisting of a set of N vertices (i.e., the data points) $\mathcal{V} = \{1, 2, ..., N\}$, along with a set of weighted edges $\mathcal{E} = \{a_{i,j}\}_{i,j \in \mathcal{V}}$, such that $a_{i,j} \geq 0$ if there is a relationship from datum j to datum i, or $a_{i,j} = 0$ otherwise. The adjacency matrix $\mathbf{A}$ of a graph is the collection of all weights, i.e., $\mathbf{A} = \{a_{i,j}\}$, $i, j = 1, ..., N$. The Laplacian matrix is defined as $\mathbf{L} = \text{diag}(\mathbf{1}^T\mathbf{A}) - \mathbf{A}$, where $\text{diag}(\boldsymbol{x})$ is a matrix having $\boldsymbol{x}$ as main diagonal, and zeros elsewhere. If the graph is undirected, the Laplacian matrix is symmetric and positive semi-definite. Also, we associate a shift operator $\mathbf{S}$ to the graph, which is an $N \times N$ matrix that encodes the sparsity pattern of the graph. Common choices for the shift operator include $\mathbf{A}$, $\mathbf{L}$, and their normalized versions [1]. A graph signal (or data) is defined as a one-to-one mapping from the set $\mathcal{V}$ of vertices to the set of real numbers. Here, we assume to have a matrix input data $\mathbf{X} := \{\boldsymbol{x}^f\}_{f=1}^{F_0} \in \mathbb{R}^{N \times F_0}$, composed of F_0 graph signals $\boldsymbol{x}^f \in \mathbb{R}^N$, $f = 1, \ldots, F_0$.

Processing tools for graph-based data hinge on the definition of graph filters [1]. In particular, an order K linear shift-invariant graph filter (LSIGF) can be written as a K-degree polynomial of the shift operator $\mathbf{S}$, with coefficients $\boldsymbol{h} = [h_0, ..., h_{K-1}]^T$. Thus, letting $\boldsymbol{u}$ and $\boldsymbol{y}$ be the input and the filtered signals, respectively, we have:

$$\boldsymbol{y} = \sum_{k=0}^{K-1} [\boldsymbol{h}]_k \mathbf{S}^k \boldsymbol{u}. \tag{1}$$

The LSIGF in (1) are able to account for the local structure of the graph, requiring information only from the K-neighborhood of each node. Thus, they represent a legit generalization of the convolution operation for signals supported on graphs [6], and are the basic building block of GCNs. The l-th layer of a GCN, taking as input $\widetilde{\mathbf{Z}}_{l-1} = \{\widetilde{\boldsymbol{z}}_{l-1}^f\}_{f=1}^{F_{l-1}}$ and yielding as output $\widetilde{\mathbf{Z}}_l = \{\widetilde{\boldsymbol{z}}_l^g\}_{g=1}^{F_l}$, with pointwise nonlinearity $\sigma_l(\cdot)$, reads as: [3]:

$$\widetilde{\boldsymbol{z}}_l^g := \sigma_l\left(\sum_{f=1}^{F_{l-1}} \sum_{k=0}^{K_l-1} [\boldsymbol{h}_l^{fg}]_k \mathbf{S}^k \widetilde{\boldsymbol{z}}_{l-1}^f\right), \quad g = 1, ..., F_l. \tag{2}$$

The order K_l of the filters, the number F_l of convolutional features of the output, and the nonlinearity $\sigma_l(\cdot)$ are hyperparameters to be chosen at each layer. Therefore, a GCN of depth L with input data $\mathbf{X}$ is built as the stack of L layers defined as in (2), where $\widetilde{\mathbf{Z}}_0 = \mathbf{X}$. Based on the learning task, an additional multi-layer perceptron (MLP) can be inserted after the last layer.

In supervised and semisupervised settings, GCNs are usually trained to learn the graph filter weights $\mathbf{H} = \{\boldsymbol{h}_l^{fg}\}_{l,f,g}$ by minimizing a (possibly regularized) task-dependent loss, e.g., cross-entropy for classification, or mean-squared error (MSE) for regression. Thus, given a training dataset $\mathcal{T}$ made of input-output pairs $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i \in \mathcal{T}}$, and a loss function $\mathcal{L}$, the learning task reads as:

$$\min_{\mathbf{H}} \quad \mathcal{L}\big(\mathbf{H}; \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i \in \mathcal{T}}\big). \tag{3}$$

In the sequel, we will show how to modify the GCN in (2) and the training strategy in (3) to incorporate tunable compression of hidden layers convolutional features, and multi-layer graph learning.

3. AUTOENCODER-AIDED GRAPH CONVOLUTIONAL NEURAL NETWORKS

The proposed AA-GCN architecture is illustrated in Fig. 1, where each layer is composed of three main stages: (i) A linear shift-invariant graph filtering stage (block (a)); (ii) Autoencoder-based compression (block (b)); (iii) Pointwise nonlinearity (block (c)). Concerning to the classical GCN model in (2), the proposed AA-GCN inserts a compression stage between the graph filtering block and the nonlinearity operations. The idea is to exploit autoencoders to perform representation learning and dimensionality reduction in the context of GCNs [15, 16]. To be more specific, an autoencoder consists of two neural networks, i.e., an encoder $f^e : \mathcal{S} \mapsto \mathcal{H}$ and a decoder $f^d : \mathcal{H} \mapsto \mathcal{S}$, where $\mathcal{H}$ is a latent space (generally) with lower dimension with respect to $\mathcal{S}$. The output of the encoder $f^e(\boldsymbol{s})$ is called the latent variable and it can be seen as a more compact, but information-rich, representation of the input signal. Typically, autoencoders are trained to minimize the MSE between the input and the reconstructed signal. Instead, in this work, our idea is to exploit them to reduce the dimension of the hidden layers' convolutional features, thus learning powerful and task-oriented low-dimensional representations in a data-driven fashion. To this aim, the layer in (2) is modified in the following way:

$$\widetilde{\boldsymbol{z}}_l^g = \sigma_l\big(\underbrace{f_l^e\big(\boldsymbol{u}_l^g\big)}_{\boldsymbol{z}_l^g}\big), \quad g = 1, ..., F_l, \tag{4}$$

where $\widetilde{\boldsymbol{z}}_l^g \in \mathbb{R}^{N_l}$, $N_l \in \mathbb{N}$ is (generally) smaller than N_{l-1}, $f_l^e : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$ is the encoder function of an autoencoder $f_l^d \circ f_l^e : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_{l-1}}$ associated with the l-th layer of the GCN, and

$$\boldsymbol{u}_l^g = \sum_{f=1}^{F_{l-1}} \sum_{k=0}^{K_l-1} [\boldsymbol{h}_l^{fg}]_k \mathbf{S}_l^k \widetilde{\boldsymbol{z}}_{l-1}^f, \quad g = 1, ..., F_l, \tag{5}$$

where $\mathbf{S}_l$ denotes the shift operator associated with the l-th layer. The compressed features at layer l are denoted as $\mathbf{Z}_l = \{\boldsymbol{z}_l^g\}_{g=1}^{F_l}$, and the final output of the layer is $\widetilde{\mathbf{Z}}_l \in \mathbb{R}^{N_l \times F_l}$. We call the stack of L layers as in (4) an Autoencoder-Aided Graph Convolutional Network (AA-GCN). Of course, based on the specific learning task, an additional MLP can be inserted after the last layer.

4. JOINT TRAINING OF AA-GCN WITH MULTI-LAYER GRAPH LEARNING

The proposed AA-GCN architecture requires a joint training of graph filter weights, autoencoder parameters, and per-layer graph representation. To this aim, let $\mathbf{A}_l$ and $\mathbf{L}_l$ be the adjacency and Laplacian matrices associated with the l-th layer, respectively. Without loss of generality, we assume that the shift operator $\mathbf{S}_l$ is a function of the adjacency $\mathbf{A}_l$. Also, let $\mathbf{W} = \{\boldsymbol{w}_l\}_{l=1}^L$ be the set of all autoencoder's parameters. Then, the joint training problem can be mathematically cast as:

$$\min_{\{\mathbf{A}_l\}_{l=1}^L, \mathbf{H}, \mathbf{W}} \quad \mathcal{L}(\{\mathbf{A}_l\}_{l=1}^L, \mathbf{H}, \mathbf{W}; \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i\in\mathcal{T}}) \tag{6}$$
$$+ \eta \sum_{l=1}^{L} \sum_{g=1}^{F_l} ||f_l^d \circ f_l^e(\boldsymbol{w}_l; \boldsymbol{u}_l^g) - \boldsymbol{u}_l^g||_2^2$$
$$+ \beta \sum_{l=1}^{L} \mathrm{Tr}\{\widetilde{\mathbf{Z}}_l^T \mathbf{L}_l \widetilde{\mathbf{Z}}_l\} - \gamma \sum_{l=1}^{L} \mathbf{1}^T \log(\mathbf{A}_l \mathbf{1}) + \lambda \sum_{l=1}^{L} ||\mathbf{A}_l||_F^2$$
subject to
$$[\mathbf{A}_l]_{i,i} = 0, \qquad [\mathbf{A}_l]_{i,j} = [\mathbf{A}_l]_{j,i} \geq 0, \qquad \forall\, i, j, l$$
$$\mathrm{Tr}\{\mathbf{L}_l\} = d_l, \quad \forall\, l$$

where λ, β, γ, η, and d_l are non-negative parameters to be tuned. The objective function of (6) is composed of the following terms: (i) The loss function $\mathcal{L}$, depending on the learning task to be performed; (ii) The autoencoders training losses, which are added to promote information-rich low-dimensional embeddings of graph convolutional features; (iii) The total variation penalties over graph signals $\widetilde{\mathbf{Z}}_l$, for all $l = 1, \ldots, L$, which promote smoothness of the encoded features over the learned graphs, as well as selective sparsity of the graph representations [17]; (iv) The logarithmic barriers $\mathbf{1}^T \log(\mathbf{A}_l \mathbf{1})$, for all $l = 1, \ldots, L$, which improve the overall connectivity of the graph and avoid the formation of isolated nodes, without compromising sparsity [18]; (v) The penalty on the squared Frobenius norms of the adjacency matrices, which helps to avoid exploding graph weights and overfitting. The constraints of Problem (6) have the following meaning: (a) There are no self-loops in the learned graphs, i.e., the main diagonals of $\mathbf{A}_l$ are null for all $l = 1, \ldots, L$; (b) The edge weights must be positive and symmetric; (c) The trace of the Laplacian of each layer is equal to a certain value d_l, to promote non-null solutions. It is interesting to notice that, if $\eta = 0$, our compression architecture reduces to simple (bottleneck) neural networks, which however loose the capacity of information-rich compression.

Since the adjacency matrices are symmetric, the number of variables of Problem (6) can be greatly reduced (approximatively by a factor of two) solving for the lower triangular parts of $\mathbf{A}_l$ for all $l = 1, \ldots, L$. In particular, let $\boldsymbol{\alpha}_l := \mathrm{vech}(\mathbf{A}_l) \in \mathbb{R}^{\frac{N(N+1)}{2}}$ be the half-vectorization of $\mathbf{A}_l$, obtained by vectorizing only the lower triangular part of $\{\mathbf{A}_l\}_l$. Then, the following relations hold:

$$\mathrm{vec}(\mathbf{A}_l) = \mathbf{M}_d \boldsymbol{\alpha}_l \quad \Longleftrightarrow \quad \mathbf{A}_l = \mathrm{vec}^{-1}(\mathbf{M}_d \boldsymbol{\alpha}_l), \tag{7}$$

where $\mathrm{vec}(\cdot)$ and $\mathrm{vec}^{-1}(\cdot)$ are the vectorization and the inverse vectorization operators, respectively, and $\mathbf{M}_d \in \mathbb{R}^{N^2 \times \frac{N(N+1)}{2}}$ is the (highly sparse) duplication matrix [19]. All the objective terms and the constraints of Problem (6) can be easily recast in terms of the variables $\boldsymbol{\alpha}_l$, for $l = 1, \ldots, L$. Solving Problem (6) enables joint training of AA-GCN parameters and multi-layer graph learning, and

Algorithm 1 : AA-GCN TRAINING

Inputs:
$\mu \in \mathbb{R}$: Learning rate.
$\Delta_\mu(\cdot)$: Optimizer-dependent backpropagation step.
$\Pi(\cdot)$: Projection operator on the feasible set of (6).
$E \in \mathbb{N}_+$: Maximum number of training iterations.
$\{\mathcal{B}_t\}_{t=1}^E$: Training dataset batches
Estimates initializations $\widehat{\mathbf{H}}_0$, $\widehat{\mathbf{W}}_0$ and $\{\widehat{\boldsymbol{\alpha}}_{l,0}\}_l$.
Loss $\mathcal{L}(\cdot)$.

Outputs:
$\{\widehat{\boldsymbol{\alpha}}_l\}_l$: Learned graph encodings.
$\widehat{\mathbf{H}}$: Learned graph filters weights.
$\widehat{\mathbf{W}}$: Learned autoencoders weights.

function AA-GCN TRAINING(**Inputs**)
  for $t \in [1, E]$ **do**
    $\widehat{\mathbf{H}}_{t+1} = \Delta_\mu\Big(\nabla_{\mathbf{H}} \mathcal{L}\ (\widehat{\mathbf{H}}_t; \mathcal{B}_t, \{\widehat{\boldsymbol{\alpha}}_{l,t}\}_l, \widehat{\mathbf{W}}_t)\Big)$
    $\widehat{\mathbf{W}}_{t+1} = \Delta_\mu\Big(\nabla_{\mathbf{W}} \mathcal{L}\ (\widehat{\mathbf{W}}_t; \mathcal{B}_t, \{\widehat{\boldsymbol{\alpha}}_{l,t}\}_l, \widehat{\mathbf{H}}_t)\Big)$
    $\widehat{\boldsymbol{\alpha}}_{l,t+1} = \Pi\Big(\Delta_\mu\Big(\nabla_{\boldsymbol{\alpha}_l} \mathcal{L}\ (\{\widehat{\boldsymbol{\alpha}}_{l,t}\}_l, \mathcal{B}_t, \widehat{\mathbf{W}}_t, \widehat{\mathbf{H}}_t)\Big)\Big),\ \forall l$
  return $\{\widehat{\boldsymbol{\alpha}}_l\}_l = \{\widehat{\boldsymbol{\alpha}}_{l,E}\}_l,\ \widehat{\mathbf{W}} = \widehat{\mathbf{W}}_E,\ \widehat{\mathbf{H}} = \widehat{\mathbf{H}}_E$

we exploit a projected descent method to achieve this task. In particular, a stochastic gradient based optimizer is chosen (e.g., SGD, ADAM, etc.), where the parameters to be learnt are properly initialized. Then, an optimizer-dependent back-propagation step is performed at each iteration on the current estimates to update them towards a descent direction. Finally, the graphs estimates are obtained by projecting the updated variables on the feasible set of (6). Interestingly, thanks to the equivalence in (7), this operation translates into a simple projection onto the simplex set [20]. In particular, denoting with $\widehat{\boldsymbol{\alpha}}_{l,t+\frac{1}{2}}$ the estimates of the optimum $\boldsymbol{\alpha}_l$ at iteration t after a backpropagation step, the steps of the projection procedure can be summarized as: (i) Set the elements of $\widehat{\boldsymbol{\alpha}}_{l,t+\frac{1}{2}}$ which will fall onto the main diagonal of the adjacency matrix $\widehat{\mathbf{A}}_{l,t+\frac{1}{2}}$ to zero, (ii) Sort the values in descending order, (iii) Set $\widehat{\boldsymbol{\alpha}}_{l,t+\frac{1}{2}} = \max\{\widehat{\boldsymbol{\alpha}}_{l,t+\frac{1}{2}} + \lambda, 0\}$, where λ is chosen such that $\mathbf{1}^T \widehat{\boldsymbol{\alpha}}_{l,t+\frac{1}{2}} = d_l$ [20], (iv) Output $\widehat{\boldsymbol{\alpha}}_{l,t+1}$ as the half-vectorization of $\widehat{\mathbf{A}}_{l,t+\frac{1}{2}}$ projected onto the feasible set of (6). Due to the sorting operation performed in step (ii), the time complexity of the projection is quasi-linear in the number of elements of $\boldsymbol{\alpha}_l$, while the other operations are linear in the same number of elements, making the training algorithm highly scalable, even for large graphs. Although the proposed algorithm is theoretically guaranteed to converge if the optimizer is SGD, several experiments validate its convergence effectiveness for general stochastic optimizers. The main steps of the proposed training procedure are listed in Algorithm 1.

5. NUMERICAL EXPERIMENTS

In this section, we first evaluate the performance of the proposed architecture and training procedure on the authorship attribution learning task, as described in [21]; then, we apply the proposed strategy to a source localization problem on a real-world graph composed of 234 Facebook users [22]. The proposed method is compared with the following state-of-the-art architectures, namely, (i) Selection GCN, (ii) Aggregation GCN (iii) Multinode GCN [3], and (iv) Graph coarsening based GCN [13]. In all the simulations, we

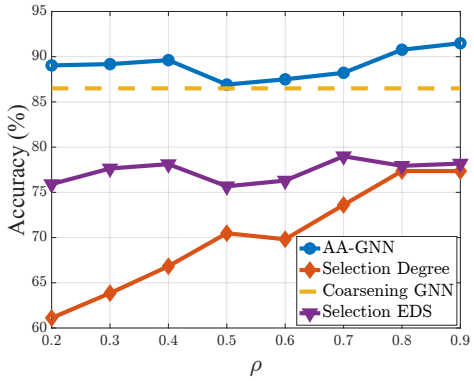

Fig. 2. Acc. vs compression ratio (ρ), authorship attribution.

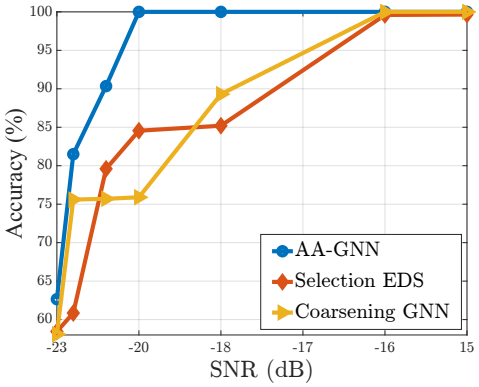

Fig. 3. Acc. vs SNR, source localization on Facebook graph.

Architecture	Accuracy
Selection Degree	65.7 ($\pm$9.12)%
Selection EDS	75.4 ($\pm$2.7)%
Selection SP	67.8 ($\pm$11.5)%
Aggregation Degree	70.5 ($\pm$2.0)%
Aggregation EDS	71.0 ($\pm$2.8)%
Aggregation SP	73.5 ($\pm$3.4)%
Multinode Degree	82.8 ($\pm$3.3)%
Multinode EDS	80.5 ($\pm$2.6)%
Multinode SP	82.2 ($\pm$2.4)%
Graph Coarsening	87.9 ($\pm$2.2)%
Autoencoder Aided	**90.3 ($\pm$2.7)%**

Table 1. Best accuracy for authorship attribution.

used an AA-GCN made of two layers as in (4) with ReLU nonlinearities, encoding functions f_l^e with one hidden layer, and a final fully-connected layer with Softmax for classification purposes. We trained our model using Algorithm 1, and exploiting the ADAM optimizer [23]. Training is performed as in (6) for AA-GCNs, and as in (3) for the competitors, where $\mathcal{L}$ is the ℓ_2 regularized cross-entropy; all the hyper-parameters are tuned via cross-validation.

1) Authorship Attribution: For the setup of this simulation, we have followed the settings of [3]. In particular, the learning task consists in deciding if an extract of a contemporary author's work from the 19$^{\text{th}}$ century is written by him/her or not. We focus on texts authored by Emily Brontë. We consider a corpus of 682 excerpts of 1000 words, written by her, taking into account $N = 211$ functional words, composing a word adjacency network (WAN) graph. We construct the dataset made of excerpts belonging to Brontë and other authors. Each sample can be interpreted as a signal defined over the WAN graph, where the value associated with each node is the frequency count of that specific functional word. For this experiment, we set $F_1 = F_2 = 32$, $K_1 = K_2 = 5$ and $N_1 = N_2 = 100$. The simulations are averaged over 10 independent dataset realizations. As a first result, in Table 1 we compare the accuracy of the proposed method against the aforementioned competitors. As we can see from Table 1, the proposed method obtains a 2% improvement with respect to the best competitor, i.e., Graph coarsening from [13]. Moreover, in Fig. 2, we illustrate the trade-off between compression and accuracy of the proposed architecture, showing the classification accuracy versus the compression factor $\rho := \frac{N_1}{N}$. The proposed AA-GCN is compared with the selection GCNs (Degree and EDS) [3] and the graph coarsening architecture [13] (taken as benchmark, due to its possibility of compressing only at $N_l = N_{l-1}/2$). From Fig. 2, we notice that the proposed architecture leads to the best performance, showing excellent robustness to compression (i.e., enhancing its scalability).

2) Source Localization: For the source localization task, the goal is to detect the community of source nodes that diffuse information over the network. We consider a real-world graph composed of 234 Facebook users [22]. In particular, we denote by $\boldsymbol{\delta}_c$ the graph signal that contains a 1 value in the position associated to a node belonging to community c, and 0 at every other node. Then, let $\boldsymbol{x} = \mathbf{A}^t\boldsymbol{\delta}_c$ be the diffused graph signal, for some unknown $t \geq 0$. Thus, given $\boldsymbol{x}$, the goal is to estimate the community c that originated the diffusion. This is clearly a classification problem in which we observe a graph signal $\boldsymbol{x}$ and have to assign it to one of the 5 communities. The hyperparamters of all the compared GCN architectures are: $F_1 = F_2 = 32$, $K_1 = K_2 = 5$, and $N_1 = N_2 = 10$. We trained the networks for 40 epochs with batches of 100 noisy training samples. Since the data realizations are random, we generate 10 different dataset realizations, and the results are averaged accross the data realizations and graph realizations. In particular, training signals are generated as $\boldsymbol{x} = \mathbf{A}^t\boldsymbol{\delta}_c + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is white Gaussian noise with variance σ_ϵ^2 (to incorporate possible model mismatching), whose components are independent from the others and from the signal. In Fig. 3, we illustrate the best classification accuracy versus the Signal to Noise ratio (SNR), defined as SNR $:= 10\log_{10}\left(\frac{\sigma_T^2}{\sigma_\epsilon^2}\right)$, where σ_T^2 is the variance of the data used for training our model. The proposed AA-GCN architecture is compared with the Selection EDS architecture from [3], and the graph coarsening strategy from [13]. As we can notice from Fig. 3, the proposed AA-GCN outperforms its competitors, illustrating its good capability to mitigate the effect of an additive noise on training data.

6. CONCLUSIONS

We introduced a novel GCN architecture, named as AA-GCN, which exploits autoencoders to enable tunable compression of the hidden convolutional features, while learning different graph representations at each layer jointly with the GCN parameters. The proposed training strategy is based on projected gradient-based optimizers, and scales well with the number of nodes of the input graph. Numerical results on both real and synthetic data illustrate the competitive performance of our method with respect to the state-of-the-art, both in terms of accuracy and robustness to compression.

7. REFERENCES

[1] D. 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 Processing Magazine*, vol. 30, 10 2012.

[2] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. on Neural Netw.*, vol. 20, pp. 61–80, 2009.

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

[4] Simone Scardapane, Indro Spinelli, and Paolo Di Lorenzo, "Distributed training of graph convolutional networks," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 87–100, 2020.

[5] Aliaksei Sandryhaila and José M. F. Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

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

[7] D. I Shuman, P. Vandergheynst, D. Kressner, and P. Frossard, "Distributed signal processing via chebyshev polynomial approximation," *IEEE Trans. on Signal and Inform. Proc. over Netw.*, vol. 4, no. 4, 2018.

[8] Y. Chen, L. Wu, and M. Zaki, "Iterative deep graph learning for graph neural networks: Better and robust node embeddings," *Proc. of NeurIPS*, vol. 33, 2020.

[9] Yu Chen, Lingfei Wu, and Mohammed J. Zaki, "Deep iterative and adaptive learning for graph neural networks," *arXiv:1912.07832*, 2019.

[10] J. Tang, W. Hu, X. Gao, and Z. Guo, "Joint learning of graph representation and node features in graph convolutional neural networks," *arXiv:1909.04931*, 2019.

[11] M. Franceschi, L.and Niepert, M. Pontil, and . He, "Learning discrete structures for graph neural networks," *Proc. of ICML*, pp. 1972–1982, 2019.

[12] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *Proc. of ICLR, Banff*, 2014.

[13] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. of Advances in Neural Information Processing Systems, Barcellona*, 2016, vol. 29.

[14] D. Mesquita, A. Souza, and S. Kaski, "Rethinking pooling in graph neural networks," *Proc. of Advances in Neural Information Processing Systems*, vol. 33, 2020.

[15] Zufan Zhang, Yunfeng Wu, Chenquan Gan, and Qingyi Zhu, "The optimally designed autoencoder network for compressed sensing," *EURASIP Journal on Image and Video Processing*, vol. 2019, no. 1, pp. 56, Apr 2019.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pp. 318–362. MIT Press, Cambridge, MA, 1986.

[17] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning laplacian matrix in smooth graph signal representations," *IEEE Transactions on Signal Processing*, vol. 64, no. 23, pp. 6160–6173, 2016.

[18] Vassilis Kalofolias, "How to learn a graph from smooth signals," *Proc. of the 19th International Conference on Artificial Intelligence and Statistics, Cadiz*, vol. 51, pp. 920–929, May 2016.

[19] Karim M. Abadir and Jan Magnus, *Matrix Algebra*, Cambridge University Press, 2005.

[20] W. Wang and M. A. Carreira-Perpinán, "Projection onto the probability simplex: An efficient algorithm with a simple proof and an application," *arXiv:1309.1541*, 2013.

[21] Santiago Segarra, Mark Eisen, and Alejandro Ribeiro, "Authorship attribution through function word adjacency networks," *IEEE Transactions on Signal Processing*, vol. 63, no. 20, pp. 5464–5478, Oct 2015.

[22] Jure Leskovec and Julian Mcauley, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. 2012, vol. 25, Curran Associates, Inc.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proc. of Int. Conference on Learning Representations, San Diego, CA, USA, May 2015*.