Applied Network Science

RESEARCH Open Access

Graph convolutional and attention models for entity classification in multilayer networks

Lorenzo Zangari[1], Roberto Interdonato[2,3], Antonio Calió[1] and Andrea Tagarelli[1*]

*Correspondence: andrea.tagarelli@unical.it
[1] Department of Computer Engineering, Modeling, Electronics, and Systems Engineering (DIMES), University of Calabria, Rende, Italy
Full list of author information is available at the end of the article

Abstract
Graph Neural Networks (GNNs) are powerful tools that are nowadays reaching state of the art performances in a plethora of different tasks such as node classification, link prediction and graph classification. A challenging aspect in this context is to redefine basic deep learning operations, such as convolution, on graph-like structures, where nodes generally have unordered neighborhoods of varying size. State-of-the-art GNN approaches such as Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs) work on *monoplex* networks only, i.e., on networks modeling a single type of relation among an homogeneous set of nodes. The aim of this work is to generalize such approaches by proposing a GNN framework for representation learning and semi-supervised classification in multilayer networks with attributed entities, and arbitrary number of layers and intra-layer and inter-layer connections between nodes. We instantiate our framework with two new formulations of GAT and GCN models, namely ML-GCN and ML-GAT, specifically devised for general, attributed multilayer networks. The proposed approaches are evaluated on an entity classification task on nine widely used real-world network datasets coming from different domains and with different structural characteristics. Results show that both our proposed ML-GAT and ML-GCN methods provide effective and efficient solutions to the problem of entity classification in multilayer attributed networks, being faster to learn and offering better accuracy than the competitors. Furthermore, results show how our methods are able to take advantage of the presence of real attributes for the entities, in addition to arbitrary inter-layer connections between the nodes in the various layers.

Keywords: Graph neural networks, Multilayer networks, Entity classification

Introduction

The topic of graph representation learning and its impact on related analysis tasks in network data has attracted great attention over the past few years, leading to one of the fastest growing subfields of research in deep learning. Graph Neural Networks (GNNs) are powerful deep learning tools that have nowadays reached state-of-the-art performances in a plethora of different tasks in graph-structured data, such as node classification, link prediction, community detection and graph classification (Xu et al. 2019). One of the main challenges addressed by these methods is to redefine basic deep learning operations, such as convolution, on structures like graph networks, where nodes may have neighborhoods that are unordered and of varying size (Bronstein et al. 2017).

The graph convolutional network (GCN) model proposed by Kipf and Welling (2017), where convolution on graphs is carried out by aggregating the values of each node's features along with its neighbors' features, paved the way for the development of further methods based on GCNs, specifically for end-to-end learning tasks or focusing on the low-dimensional embedding generation; in the latter case, for instance, the graph autoencoder (GAE) model (Kipf and Welling 2016) is one of the earliest approaches for unsupervised learning, clustering and link prediction on graphs based on GCNs. Another important advance in deep learning, namely the attention mechanism, has also inspired several studies that apply it to graph-structured data. The graph attention network model (GAT) by Velickovic et al. (2018) exploits a masked self-attention mechanism in order to learn weights between each couple of connected nodes, where self-attention allows for discovering the most representative parts of the input. It should be noted that the above methods work on simple networks only, i.e., networks modeling a single type of relation for a homogeneous set of nodes. Due to the growing interest for modeling and mining multilayer networks (Kivelä et al. 2014) as well as for developing network embedding models for attributed or feature-rich networks (Gaito et al. 2021; Interdonato et al. 2019), in the last few years a significant effort has been put in studying representation learning models and techniques specifically conceived for multilayer networks with associated attribute information at node level. Nevertheless, representation learning becomes even more challenging for multilayer networks because of the presence of intra-layer and inter-layer relations, different layer characteristics, as well as node features. In particular, these approaches must be able to obtain new latent node representations based on intra- and inter-layer dependencies. For example, in a cross-platform multilayer network one might want to predict a user's gender based on the relationships and properties that users and relating contacts have on each platform.

A further source of complexity also relates to the partial knowledge that we may have about the attributes associated with the nodes in one or multiple layers of the multilayer network in input. Moreover, for the purpose of a classification or prediction task, it is often the case in a real scenario that the labels for a given target concept (i.e., class) are available for few nodes only.

The aim of this work is to revise main GNN approaches to address both representation learning and prediction problems in a multilayer network. Our focus is on how to model properties of the multilayer network topology and of the available node attributes, so to learn an effective representation of the node features within and across the multiple layers of the input network. Furthermore, the aim is also to exploit the learned representations to predict the class labels of the entities (or actors) in the network. A key requirement in our proposed approach is also its flexibility to multilayer networks characterized not only by arbitrary node-coverage and number of layers, but also by the presence of inter-layer relations that are not constrained to link node instances of the same entity over the layers.

We summarize our contributions as follows:

1. To cope with partial knowledge on attributes as well as class labels at node-level, which is usually encountered in real scenarios, we address the end-to-end learning problem of embedding and classification for the entities in a multilayer attributed

network following a *transductive semi-supervised* learning approach. In this context, class labels are known at training time only for a relatively small amount of nodes in the multilayer network, while all available structural information and node attributes can be exploited for learning, and the goal is to predict the labels of the unlabeled nodes.

2. We propose a representation learning and node classification framework based on GNN models and designed for arbitrary multilayer attributed networks. In accord with the significant trend in literature whereby graph convolutional and attention-based approaches are by far the most widely used, the core GNN component of our framework is instantiated both as GCN and GAT.
3. Unlike existing GNN approaches for multiplex or multirelational graphs, we propose to aggregate topological neighborhood information from different layers directly into the propagation rule of the GNN component, i.e., during its forward learning phase, in order to make the embedding of an entity in a particular layer depending on both its neighbors in that layer (dubbed *within-layer* neighborhood) and on its neighbors located in other layers where the entity occurs (referred to as *outside-layer* neighborhood). Therefore, by K sequential applications of our multilayer designed GNN components, the K-hop within-layer and outside-layer neighborhood structural information for each entity is incorporated in the embedding process.
4. Our designed GNN components in the proposed framework are able to incorporate external information associated with the multilayer network, in the form of attributes that can be available at entity-level or at node-level for each particular layer of the input network.
5. Experimental evidence from widely used multiplex networks and from a real-world attributed multilayer network dataset has shown that both the GAT and the GCN instances of our framework represent effective and efficient solutions to the problem of entity classification in multilayer attributed networks. Our methods were also compared with two recently proposed methods for multirelational networks based on a GAT model, named GrAMME-SG and GrAMME-Fusion (Shanthamallu et al. 2020): our methods are able to achieve accuracy as good as or better than the competitors (up to 13% of accuracy improvement), while outperforming them in terms of efficiency (with a training time which is two orders of magnitude lower than GrAMME methods in most cases).

Background

In this section we provide background notions on deep learning approaches based on Graph Neural Networks (GNNs). For a better comprehension, we first introduce some preliminary notations.

Preliminary notations. We are given a graph $G=(V,E)$, where V is the set of nodes, with $|V|=n$, and E is the set of edges. Besides the adjacency matrix $\mathbf{A}$ that represents the graph structure, a further matrix is provided in input, $\mathbf{X}$, which stores the feature descriptions of the nodes, i.e., each node v_i is provided with a vector $\mathbf{x}_i \in \mathbb{R}^f$, where f is the number of input features. The general goal for GNNs is to learn a function that takes in input the above matrices and yields an output feature representation of the nodes

Table 1 Summary of notations and their description

Notations	Description
G	A simple (i.e., monoplex) network graph
V, E	Set of nodes and set of edges in G
$G_{\mathcal{L}}$	A multilayer network graph
$\mathcal{V}$	Set of entities in $G_{\mathcal{L}}$
$\mathcal{L}, \ell, L_l$	Set of layers, number of layers, l-th layer of $G_{\mathcal{L}}$
$V_{\mathcal{L}}, E_{\mathcal{L}}$	Set of nodes and set of edges in $G_{\mathcal{L}}$
i	Index of node v_i in G, resp. entity v_i in $G_{\mathcal{L}}$
$\Gamma(i)$	Neighborhood of node v_i in G
$\Gamma(i,l)$	Within-layer neighborhood of v_i in layer L_l of $G_{\mathcal{L}}$
$\Psi(i,l)$	Outside-layer neighborhood of v_i in layers of $G_{\mathcal{L}}$ different from L_l
$\mathbf{X}, \mathbf{X}_l$	Attribute (input feature) matrix in G, resp. in the l-th layer of $G_{\mathcal{L}}$
$\mathbf{x}_i, \mathbf{x}_{(i,l)}$	Attribute (input feature) vector for node v_i in G, resp. entity v_i in layer L_l of $G_{\mathcal{L}}$
$\mathbf{Z}, \mathbf{Z}_l$	Embedding (output feature) matrix in G, resp. in the l-th layer of $G_{\mathcal{L}}$
$\mathbf{z}_i$	Embedding (output feature) vector for node v_i
$\widetilde{\mathbf{Z}}$	Embedding (output feature) matrix for the entities in $G_{\mathcal{L}}$
$\mathbf{h}_i$	Hidden-layer vector for node v_i
$\mathbf{h}^{(k)}_{(i,l)}$	Hidden-layer vector at the k-th layer of the GNN for entity v_i in layer L_l of $G_{\mathcal{L}}$
f	Number of attributes (input features)
d	Size of the embedding
K, k	Number of GNN layers, index of layer
Q, q	Number of attention heads, index of attention head
$\mathbf{W}, \mathbf{W}^{(k)}$	Weight matrix of a generic, resp. k-th, layer of a GNN
$\mathbf{A}, \mathbf{A}_l$	Adjacency matrix in G, resp. in the l-th layer of $G_{\mathcal{L}}$
$\mathbf{A}^{\text{sup}}$	Supra-adjacency matrix in $G_{\mathcal{L}}$
$\widetilde{\mathbf{A}}, \widetilde{\mathbf{A}}^{\text{sup}}$	Adjacency matrix, resp. supra-adjacency matrix, with self-loops
$\sigma(\cdot)$	Activation function
e_{ij}	Attention coefficient for edge between nodes v_i and v_j
α_{ij}	Normalized attention coefficient for edge between nodes v_i and v_j

$\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_n]^{\mathrm{T}}$, where $\mathbf{z}_i \in \mathbb{R}^d$ denotes the *embedding* or output feature vector for node $v_i \in V$ and d is the size of the embedding space. Every GNN layer can be modeled as a non-linear function $\mathbf{H}^{(k+1)} = g(\mathbf{H}^{(k)}, \mathbf{A})$, where k is an index for a neural network layer, $\mathbf{H}^{(0)} = \mathbf{X}$ and $\mathbf{H}^{(K)} = \mathbf{Z}$, with K total number of neural network layers. Table 1 summarizes the main notations that will be used throughout this paper.

Deep graph learning. Deep learning frameworks such as convolutional neural networks (CNNs) (LeCun and Bengio 1995), recurrent neural networks (RNNs) (Hochreiter and Schmidhuber 1997) and autoencoders (AEs) (Vincent et al. 2010) have been extremely successful in several machine learning tasks and for a variety of domains, including grid-structured data (e.g., images), sequences, and text data (LeCun et al. 2015). However, they cannot be straightforwardly applied to graph-structured data as well because several operations (e.g., convolution) need to be revised to be well suited for such type of data. Indeed, graph-structured data show complexities at different and more levels than other types of data, which include the lack of natural orderings of the nodes and/or edges, variability in the size and topology of a node's neighborhood, and the opportunity for modeling different types of node relations (Wu et al. 2021).

In recent years, the trend of using deep learning techniques to analyze graphs has contributed to the birth of connectionist models called Graph Neural Networks (GNNs), which aim to extend deep learning on graph-structured data, exploiting its dependencies through a message passing scheme between the nodes (Zhou et al. 2019). In contrast to random walk approaches, which consider only nodes co-occurring in a random walk and optimize the embeddings to encode random walk statistics (Grover and Leskovec 2016; Perozzi et al. 2014), GNN carries out a scheme for which each node iteratively combines both the neighbors and its own features to obtain a new representation. After k iterations (i.e., k-th layer of the GNN), node representations have a non-linear relation with their k-hop away neighborhood information. Interestingly, the neighborhood aggregation scheme is strictly connected to a random walk process: in fact, as studied in Xu et al. (2018), in a K-layer GNN, the influence distribution of node v_i (i.e., how much a change in the initial features of any node v_j affects the final embedding of v_i) is equivalent, in expectation, to a random walk of length K starting from node v_i, therefore the influence of v_i by v_j is proportional to the probability of visiting node v_j in a random walk of length K starting from node v_i. Moreover, it should be noted that using a high number of iterations (i.e., K) could lead to an *over-smoothing* problem, i.e., representations of nodes could become very similar to one another after several iterations, as this would be an effect of an overly expanded range of the node influences.

GNNs are end-to-end trainable, i.e., they can be trained in a supervised or unsupervised manner depending on the task to be performed, and they are designed to compute the new embedding state using both structural information of the graph and properties of nodes and edges, through an iterative neighborhood properties aggregation scheme. This final embedding state can be used to produce an output such as the node labels, or even to obtain the representation of an entire graph through pooling, for example, by summing the representation vectors of all nodes in the graph (Xu et al. 2019).

Two of the most successful approaches are Graph Convolutional Networks (GCN) (Kipf and Welling 2016) and Graph Attention Networks (GAT) (Velickovic et al. 2018), both convolutional-style GNNs, but with different assumptions regarding the contribution of the neighborhood. More specifically, the former model adopts a spectral approach in which the convolution is defined by signal theory filters, while GAT aims to incorporate the attention mechanism in the propagation step, learning the importance of the neighborhood of each node through a masked self-attention strategy. In the following, we briefly review the above two approaches.

Graph convolutional network. A GCN is the counterpart of a convolutional neural network model for graph-structured data that uses a graph spectral approach to convolution. Specifically, it operates through a first-order spectral approximation of the graph by restricting the filters (limiting the order of the Chebyshev polynomial) to operate in the neighborhood at one step away from each node.

Equation (1) shows the propagation rule of a GCN layer, which is the building block of the model, as it aims to learn a function capable of generating new feature representations for each node $v_i \in V$ by propagating and transforming its own features and those of its neighbors:

$$\mathbf{z}_i = \sigma\Big(\sum_{j\in\Gamma(i)\cup\{i\}} \frac{1}{\sqrt{\widetilde{\mathbf{D}}_{ii}\widetilde{\mathbf{D}}_{jj}}}\mathbf{h}_j\mathbf{W}\Big). \tag{1}$$

Above, $\Gamma(i)$ denotes the set of neighbors of node i, $\sigma(\cdot)$ is a non-linear activation function (e.g., $ReLU(\cdot) = max(0,\cdot)$), $\mathbf{W}$ is a layer-specific trainable weight matrix, and $\widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{ij}$ is the degree matrix derived from $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n$, where $\mathbf{A}$ is the adjacency matrix of the input graph G, and $\mathbf{I}_n$ is the identity matrix of size n.

Note that self-loops are added to the graph, and the adjacency and degree matrices are built accordingly (this is known as *renormalization trick*), so that each node can also consider its own features, and potential numerical issues can be controlled; specifically, symmetric normalization is used because repeated applications of the propagation rule can lead to numerical instability and problems in the calculation of the gradient when used in the deep neural network.

GCN plays a central role in building many complex models of GNNs, which also includes unsupervised learning architectures such as the Graph Auto-Encoders (GAEs) (Kipf and Welling 2016; Zhou et al. 2019). A GAE is a framework for unsupervised learning that leverages GCN to encode node information into low-dimensional vectors. Specifically, the encoder that calculates the embeddings consists of two GCN layers with a non-linear activation function, whereas the decoder to reconstruct the original adjacency matrix from node embeddings, is a simple inner product decoder. The model is hence trained by minimizing the similarity between the original adjacency matrix and the reconstructed one.

Graph attention network. Graph Attention Network (GAT) (Velickovic et al. 2018) is a graph neural network architecture that uses the attention mechanism to learn weights between connected nodes. In contrast to GCN, which uses predetermined weights for the neighbors of a node corresponding to the normalization coefficients described in Eq. (1), GAT modifies the aggregation process of GCN by learning the strength of the connection between neighboring nodes through the attention mechanism (Wu et al. 2021).

The building block is a Graph Attention Layer (GAT layer) which generalizes the attention model on graph structured data and is agnostic of the particular choice of attention mechanism. Stacking GAT layers several times allows one to develop deep neural network architectures.

In order to learn the weighting factor of each node's features, *attention coefficients* are computed based on the features of the connected nodes using a function $a : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$. Equation (2) indicates the importance of node j's features to node i:

$$e_{ij} = a(\mathbf{h}_i, \mathbf{h}_j). \tag{2}$$

The graph structure is taken into account as for each node v_i, its neighborhood is considered in performing a masked attention mechanism. In Velickovic et al. (2018) the attention mechanism is a feed-forward neural network, which utilizes the non-linear activation function *LeakyReLU*: instead of outputting a 0 for all negative values as *ReLU* does, *LeakyReLU* outputs a value of $-\beta x$ for any input x that is negative (where β is an hyperparameter that determines the amount of leak, usually set between 0.01 and 0.2), whereas for positive values of x, it simply outputs x. By allowing non-zero gradient for negative values, *LeakyReLU* overcomes the issue of dying neurons that affects the *ReLU* function.

Then, the attention coefficients are normalized through *Softmax* function as in Eq. (3), so that the attention weights sum up to 1 over all neighbors of a node, eventually obtaining the normalized attention coefficients α:

$$\alpha_{ij} = \frac{\exp(LeakyReLU(e_{ij}))}{\sum_{t \in \Gamma(i)} \exp(LeakyReLU(e_{it}))}. \tag{3}$$

As next step, each node updates its hidden state by weighting the features of the neighborhood nodes with the attention coefficients according to Eq. (4):

$$\mathbf{z}_i = \sigma\Big(\sum_{j \in \Gamma(i)} \alpha_{ij} \mathbf{W} \mathbf{h}_j\Big). \tag{4}$$

Note that the learnable weight matrix $\mathbf{W}$ is pre-applied to every node in order to transform the input features into higher-level features.

To stabilize the learning process of self-attention, the mechanism has been extended similarly to Vaswani et al. (2017) by employing *multi-head* attention. The operations of the layer are independently replicated Q times (with different parameters) and outputs are feature-wise aggregated. Equation (5) shows the computation of a linear combination of the features by concatenating the Q attention heads, where $\alpha_{ij}^{(q)}$ is normalized attention coefficient computed by the q-th attention mechanism:

$$\mathbf{z}_i = \underset{q=1\ldots Q}{\Big|\Big|} \sigma\Big(\sum_{j \in \Gamma(i)} \alpha_{ij}^{(q)} \mathbf{W}^{(q)} \mathbf{h}_j\Big), \tag{5}$$

where $\|$ denotes the concatenation operator. For the combination of the Q independent heads, Velickovic et al. (2018) suggests to concatenate them in the hidden layers and to average them in the final layer; in the latter case, the application of the σ function is delayed.

Limitations of GNNs. Although GNNs can exploit node attributes and topology of the input graph, their learning power for a node classification task could be limited when there is a misaligment between features, graph and class label. While combining graph and feature information generally leads to an improvement in classification performance, the study in Qian et al. (2021) has shown the importance of graph and feature alignment in GNN models such as GCN, highlighting that when features and graph subspaces associated with the data are not aligned, the GCN approach can exhibit a performance degradation, being even outperformed by an MLP model learned from data features while discarding the network topology. More specifically, if the node connections in the network are not consistent with the associated node-features (e.g., two adjacent nodes having significantly different features), then the node-neighborhood aggregation scheme could not be beneficial. As a matter of fact, typical schemes of neighborhood aggregation in GNNs inherently assume the homophily principle, i.e., connected nodes have the same class label or similar features. Another study proposed in Zhu et al. (2020) has shown that learning on networks with low homophily (i.e., connected nodes have different class labels) is a challenging task for GNNs, which could perform worse than

MLP. However, as reported in other studies, such as Ma et al. (2021), GCN models can still achieve good performance on low homophily networks, provided that nodes with the same class have similar neighborhoods, and different classes have distinguishable patterns.

Although investigating the aforementioned limitations is not a focus of this work, we shall take into account them in our experimental evaluation. In particular, we measure the homophily score of evaluation networks (cf. "Data" section), and we investigate on the impact of not using real-world features for our evaluation networks, where class assignment is based exclusively on graph topology (cf. "Evaluation with competing methods" section).

Related work

In order to contextualize our proposal with respect to existing literature, we here discuss some of the recently proposed GNN methodologies specifically designed for multilayer networks.

One of the first frameworks that considers inter-layer edges for embedded representation learning is MANE (Li et al. 2018). However, the optimization problem of node embedding solved in MANE does not account for node attributes, and its overall approach is not end-to-end. By contrast, approaches that aim to extend deep-learning based methods for single-layer graphs such as GCN and GAT are well-suited for modeling both within- and inter-layer dependencies to generate embeddings for nodes that are associated with input features, and in addition they have the advantage of being designed to learn node embedding and a classifier simultaneously via an end-to-end approach. Indeed, node classification approaches for multilayer networks have been recently proposed.

Ghorbani et al. (2019) proposed *MGCN* to extend the GCN model to multilayer networks. The method builds a GCN for each layer of the network, by utilizing only links between nodes of the same layer, while discarding the inter-layer relations. To consider inter-layer dependencies, the method uses an unsupervised term in the loss function, which calculates the ability of reconstructing the network through the inner product of the embeddings. Our proposed method shares with MGCN the design for solving a semi-supervised classification problem where label information is smoothed over the graph structure via regularization, according to Kipf and Welling (2017). However, unlike MGCN, our proposed ML-GCN method is able to incorporate the inter-layer edges within the GCN propagation rules, as well as in the loss function.

While MGCN is an extension of GCN, the GrAMME method in Shanthamallu et al. (2020) extends GAT for multilayer networks. The peculiarity of this approach is the way the Q attention heads are combined: instead of concatenating or averaging them as suggested in Velickovic et al. (2018), in GrAMME a mechanism called *fusion-head* is applied, which consists in a weighted combination of the attention heads with learnable parameters. Specifically, in Shanthamallu et al. (2020) two approaches are developed, namely GrAMME-SG and GrAMME-Fusion. The former explicitly builds the inter-layer edges between each node in a layer and its counterpart (referred to as *pillar edges*) in a different layer, and applies a series of GAT layers with the fusion-head method, exploiting the inter-layer dependencies. The GrAMME-Fusion approach deals with inter-layer dependencies in a different way, as it builds layer-wise attention models and introduces an additional layer that exploits inter-layer dependencies using only fusion heads. In the case of nodes with missing attributes, both methods employ random initialization (using a standard normal distribution). The empirical evaluation reported by the authors with several multiplex networks showed that the GrAMME-Fusion method performs better than GrAMME-SG.

Our proposed GAT extension to multilayer networks shares the multi-head attention mechanism with the GrAMME methods, although our approach is closer to GAT as it does not need the fusion-head strategy to integrate the inter-layer dependencies. More importantly, our approach involves both within-layer and outside-layer neighborhoods when computing the embedding of an entity in each layer, while GrAMME-SG involves only pillar edges (in addition to the local neighborhood) in the propagation rule, and GrAMME-Fusion integrates the inter-layer dependencies using only fusion heads. Note that, given its declared superiority with respect to convolutional approaches, in our experimental evaluation, we have referred to the GrAMME methods as main competitors of our proposed methods.

Proposed framework

Given a set $\mathcal{V}$ of N *entities* (e.g., users) and a set $\mathcal{L} = \{L_1, \cdots, L_\ell\}$ of *layers* (e.g., user relational contexts), with $|\mathcal{L}| = \ell \geq 2$, we denote a multilayer network with $G_\mathcal{L} = \langle V_\mathcal{L}, E_\mathcal{L}, \mathcal{V}, \mathcal{L}\rangle$, where $V_\mathcal{L} \subseteq \mathcal{V} \times \mathcal{L}$ is the set of entity-layer pairings or *nodes* (i.e., to denote which users are present in which layers), and $E_\mathcal{L} \subseteq V_\mathcal{L} \times V_\mathcal{L}$ is the set of undirected edges between nodes within and across layers.[1]

We represent a multilayer network by a set of adjacency matrices $\mathcal{A} = \{\mathbf{A}_1, \cdots, \mathbf{A}_\ell\}$, with $\mathbf{A}_l \in \mathbb{R}^{n_l \times n_l}$ ($l = 1..\ell$), where $n_l = |V_l|$. Entities are assumed to be associated with *features* stored in layer-specific matrices $\mathcal{X} = \{\mathbf{X}_1, \cdots, \mathbf{X}_\ell\}$, with $\mathbf{X}_l \in \mathbb{R}^{n_l \times f_l}$ and f_l the number of node features in the l-th layer. We will also use symbol $\mathbf{x}_{(i,l)}$ to denote the feature vector of entity v_i in layer L_l.

Note that in our multilayer network model there is neither prior assumption about the set of valid couplings between the layers, nor about the structure of the layers. Indeed our framework is theoretically able to consider networks with different coupling constraints between the layers, e.g. temporal networks or cross-platform networks.

It is also worth noticing that the sizes f_l may differ in principle, however they all must be bounded with respect to a maximum size, say f; truncation, resp. zero-padding, apply for those layers having a greater, resp. lower, number of features than f. Moreover, to avoid numerical scaling issues, all feature matrices are assumed to be row-normalized within a common interval of values. Furthermore, in case no node attributes are available for $G_\mathcal{L}$, each layer-specific feature matrix is assumed to be the identity matrix $\mathbf{I}_l \in \mathbb{R}^{n_l \times n_l}$. Also, for partially complete feature matrices, value imputation and matrix completion methods can certainly be used, however this goes beyond the scope of this work.

[1] In this work we will use the term *layer* to denote either a constituent of a multilayer network or a constituent of the neural network model, while the exact meaning is assumed to be clear as within the particular context where the term is used.

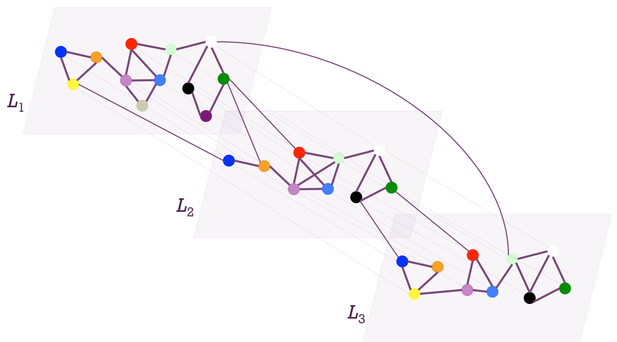


Fig. 1 Example multilayer network handled by our framework, with color codes mapping nodes of the same entity: in addition to within-layer edges (solid thick lines), note the presence of both inter-layer pillar edges (dashed thin lines) and inter-layer edges connecting nodes of different entities (solid thin lines)

Node embedding in multilayer network. Given a multilayer network $G_{\mathcal{L}} = \langle V_{\mathcal{L}}, E_{\mathcal{L}}, \mathcal{V}, \mathcal{L} \rangle$, we define the multilayer network embedding as the problem of learning low-dimensional latent representations for each node (i.e., entity-layer pair), that is, learning a function $g : V_{\mathcal{L}} \mapsto \mathbb{R}^d$ that maps each node into a d-dimensional space, with $d \ll N$, so that nodes that are similar in $G_{\mathcal{L}}$ have embeddings close to each other.

The above definition resembles the classic one of node embeddings, with adaptation to multilayer networks. Moreover, to model similarity of nodes in the multilayer network, we follow the general idea adopted in representation learning on graphs, that is, node embeddings are generated based on neighborhoods, upon the intuition that nodes aggregate information from their neighbors by using a GNN.

However, a major question becomes how to consider a node's neighborhood in the multilayer network to properly generate the embeddings. In this regard, we notice that a major requirement for our proposed framework is to account for node links that are internal as well as external to a particular layer where the nodes occur. To this purpose, our key idea is to incorporate in the GNN propagation rules aggregation over node features—both topological and exogenous to the network, i.e., node attributes—that are computed not only w.r.t. the node's neighbors in the same layer but also w.r.t. the node's neighbors in the other layers.

In this regard, we define two functions, denoted as Γ and Ψ, that for each pair entity-layer, i.e., node, return the neighborhood of the entity that is internal and external to that layer, respectively. Formally, given an entity v_i in a layer L_l, we define the set of *within-layer* neighbors of v_i in layer L_l as:

$$\Gamma(i,l) = \{(j,l) \in V_{\mathcal{L}} \mid ((j,l),(i,l)) \in E_{\mathcal{L}}\}. \tag{6}$$

Similarly, we define the set of *outside-layer* neighbors of v_i in layers different from L_l as:

$$\Psi(i,l) = \{(j,m) \in V_{\mathcal{L}} \mid ((j,m),(i,l)) \in E_{\mathcal{L}}, m \neq l\}. \tag{7}$$

Figure 1 shows an illustration of multilayer network that our framework is able to deal with: in fact, more generally than multiplex networks, inter-layer edges can be formed to link not only nodes of the same entity but also nodes of different entities. Both types of inter-layer edges are indeed considered in our definition of outside-layer neighbors (cf. Eq. 7).

Let us now consider the key constituents in our proposed GNN models, precisely a GCN model and a GAT model for multilayer networks. First, we denote with $\mathbf{h}_{(i,l)}^{(k)}$ the hidden state at the k-th layer of the neural network for entity v_i in layer L_l, and with $\mathbf{z}_{(i,l)} = \mathbf{h}_{(i,l)}^{(K)}$ the final embedding of entity v_i in L_l, eventually used for a downstream task, such as entity classification. Using the message passing paradigm (Gilmer et al. 2017), we abstract the aggregation scheme of our framework in Eq. (8):

$$\mathbf{h}_{(i,l)}^{(k+1)} = \phi_v\left(\mathbf{h}_{(i,l)}^{(k)}, \bigoplus_{(j,m)\in\Gamma(i,l)\cup\Psi(i,l)} \phi_e(\mathbf{x}_e, \mathbf{h}_{(i,l)}^{(k)}, \mathbf{h}_{(j,m)}^{(k)})\right), \tag{8}$$

where the ϕ_e function, named *message function*, is edge-wise defined to generate messages across the edges obtained by combining the edge properties $\mathbf{x}_e$, and the state of its two end-nodes, i.e., $\mathbf{h}_{(i,l)}$ and $\mathbf{h}_{(j,m)}$; ϕ_v, named *update function*, is a node-wise function useful to update the state of a node; $\bigoplus$ is the aggregation (or reduce) operator, which is usually summation, or alternatively a pooling operator or even a neural network (Wang et al. 2020). Note that in our formulation a node updates its state considering both intra-layer and inter-layer dependencies within the aggregation stage, so that the embedding of each layer is related to the other layers. If the framework is instantiated based on a GAT approach, the message ϕ_e of each edge (i, l), (j, m) received by node v_i corresponds to the normalized attention coefficient $\alpha_{(i,l),(j,m)}$ multiplied by the hidden state of node v_j.

Our contribution is to incorporate the above representation models into the propagation rules of both GCN and GAT frameworks, in order to make them suitable for the multilayer network context. Our resulting methods are dubbed ML-GCN and ML-GAT, respectively.

ML-GCN *propagation rules.* Given a node v_i in a layer L_l, the first propagation rule is defined as:

$$\mathbf{h}_{(i,l)}^{(1)} = \sigma\left(\sum_{(j,m)\in\Gamma(i,l)\cup\Psi(i,l)} \frac{1}{\sqrt{\widetilde{\mathbf{D}}_{ii}\widetilde{\mathbf{D}}_{jj}}} \mathbf{W}^{(0)^{\mathrm{T}}} \mathbf{h}_{(j,m)}^{(0)}\right), \tag{9}$$

where $\mathbf{h}_{(i,l)}^{(0)} = \mathbf{x}_{(i,l)}$, $\sigma(\cdot)$ is the ReLU activation function, and $\mathbf{W}^{(0)}$ is the initial weight matrix of shape (f, d), shared across all nodes of the multilayer graph. Note that the degree matrix $\widetilde{\mathbf{D}}$ is built considering both inter-layer and intra-layer connections of nodes using the *supra-adjacency* matrix of the graph, which can be defined as:

$$\mathbf{A}^{\text{sup}} = \begin{cases} \mathbf{A}_l \in \mathcal{A} & \text{if diagonal block} \\ \mathbf{A}_{l,m} & \text{otherwise (i.e., off-diagonal block)} \end{cases}, \tag{10}$$

where $A_{l,m}$ is an inter-layer adjacency matrix built upon the inter-layer connections between layer l and layer m (i.e., 1 if there exists an edge between (i, l) and (u, m) with $l \neq m$, and 0 otherwise). Moreover, $\widetilde{\mathbf{D}}_{ii} = \sum_{j=1} \widetilde{\mathbf{A}}_{ij}^{\text{sup}}$, where $\widetilde{\mathbf{A}}^{\text{sup}}$ is the supra-adjacency matrix with self-loops added.

The above equation is then adapted to produce the propagation rule at the generic k-th layer of the GNN (with $1 \leq k < K$):

$$\mathbf{h}_{(i,l)}^{(k+1)} = \sigma\left(\sum_{(j,m)\in\Gamma(i,l)\cup\Psi(i,l)} \frac{1}{\sqrt{\widetilde{\mathbf{D}}_{ii}\widetilde{\mathbf{D}}_{jj}}}\mathbf{W}^{(k)^{\mathrm{T}}}\mathbf{h}_{(j,m)}^{(k)}\right). \tag{11}$$

Note that, for $k = K - 1$, the above equation produces the output feature vector for entity v_i in layer L_l. Also, the weight matrix $\mathbf{W}^{(k)}$ shape is (d, d), for every $k \geq 1$.

ML-GAT *propagation rules.* Given a node v_i in a layer L_l, the first propagation rule is defined as:

$$\mathbf{h}_{(i,l)}^{(1)} = \sigma\left(\sum_{(j,m)\in\Gamma(i,l)\cup\Psi(i,l)} \alpha_{(i,l),(j,m)}\mathbf{W}^{(0)^{\mathrm{T}}}\mathbf{h}_{(j,m)}^{(0)}\right), \tag{12}$$

where $\alpha_{((i,l),(j,m))}$ is the normalized attention coefficient for any edge $((i,l),(j,m)) \in E_{\mathcal{L}}$. Note that we integrate the attention mechanism both on intra-layer and inter-layer edges, so that our model can selectively integrate the information received from the inter-layer and intra-layer neighbors.

Similarly to the solution proposed for ML-GCN, the initial propagation rule equation is generalized for any k-th layer of the GNN (with $1 \leq k < K$). In addition, a mechanism of multi-head attention is used to stabilize the learning process of self-attention. Therefore, given Q attention heads, we define two variants of the generic propagation rule, where either the output features of the heads are concatenated:

$$\mathbf{h}_{(i,l)}^{(k+1)} = \Big\|_{q=1\ldots Q}\, \sigma\left(\sum_{(j,m)\in\Gamma(i,l)\cup\Psi(i,l)} \alpha_{(i,l),(j,m)}^{(q)}\mathbf{W}^{(q,k)^{\mathrm{T}}}\mathbf{h}_{(j,m)}^{(q,k)}\right), \tag{13}$$

or the heads are averaged before applying the activation function:

$$\mathbf{h}_{(i,l)}^{(k+1)} = \sigma\left(\frac{1}{Q}\sum_{q=1\ldots Q}\left(\sum_{(j,m)\in\Gamma(i,l)\cup\Psi(i,l)} \alpha_{(i,l),(j,m)}^{(q)}\mathbf{W}^{(q,k)^{\mathrm{T}}}\mathbf{h}_{(j,m)}^{(q,k)}\right)\right). \tag{14}$$

Above, $\mathbf{h}_{(j,m)}^{(q,k)}$ denote the embedding of node (j, m) for the q-th head of the k-th layer of the neural network. Moreover, in our setting, we averaged the Q attention heads in order to save memory occupation; therefore we set σ as the exponential linear unit (*ELU*) activation function, i.e., for positive values of input b, the function simply outputs b, whereas if the input is negative, the output is $exp(b) - 1$.

Entity classification. As downstream task, we consider the problem of classifying the entities of a multilayer network graph in a *transductive* setting, i.e., class labels are only available for a small subset of entities, however the whole network graph containing both labeled and unlabeled data is used during the learning process, and the goal of the trained model is to predict the labels of the unlabeled entities. As previously discussed in "Introduction" section, this setting complies with the realistic assumption of lack of knowledge on a target concept corresponding to the class, for most of the nodes in a network. However, the transductive setting is also challenging as it requires that our

GNN-based learning framework must be able to learn representation not only of nodes with labels but also of nodes without labels. We define this type of (multiclass) classification task with partial supervision, i.e., semi-supervised classification task, as follows:

Problem 1 (Entity classification in multilayer network) *Given a multilayer network $G_{\mathcal{L}} = \langle V_{\mathcal{L}}, E_{\mathcal{L}}, \mathcal{V}, \mathcal{L}\rangle$, and associated input feature matrix $\mathcal{X}$, let $\mathbf{Y} \in \mathbb{R}^{N\times C}$ denote the binary matrix storing the class labels assigned to each entity in $\mathcal{V}$, where C is the number of a predetermined set of classes. Given a small subset of entities $\mathcal{V}_{train} \subset \mathcal{V}$ that we refer to as training set, we denote as $\mathbf{Y}_{train} \in \mathbb{R}^{|\mathcal{V}_{train}|\times C}$ the corresponding class label matrix. The goal is to predict the label of the entities in $\mathcal{V}\backslash\mathcal{V}_{train}$ using both the multilayer graph structure based on the supra-adjacency matrix and the entity features stored in $\mathcal{X}$. That is, we want to obtain the probability distribution matrix $\widehat{\mathbf{Y}} \in \mathbb{R}^{N\times C}$ so to derive $\widehat{\mathcal{Y}} = \arg\max_c \widehat{\mathbf{Y}}$ that assigns class labels to the entities in $\mathcal{V}\backslash\mathcal{V}_{train}$.*

Algorithm 1: Our proposed framework for node embedding generation and entity classification in a multilayer network

```
Data: $G_{\mathcal{L}} = \langle V_{\mathcal{L}}, E_{\mathcal{L}}, \mathcal{V}, \mathcal{L}\rangle$ multilayer network (with $\ell = |\mathcal{L}|$); $\mathbf{Y}_{train}$ true class labels for the entities in
      $\mathcal{V}_{train} \subset \mathcal{V}$; $\mathcal{X} = \{\mathbf{X}_1, \cdots, \mathbf{X}_\ell\}$ layer-specific input feature matrices; $K$ number of hidden layers; $T$
      number of epochs
Output: Entity prediction $\widehat{\mathcal{Y}}$
1  Build supra-adjacency matrix $\mathbf{A}^{\text{sup}}$ (cf. Eq. 10)
2  Apply Glorot initialization to all parameters
3  for $t = 1 \dots T$ do
4      $\mathbf{H}^{(0)} = \mathcal{X}$
5      for $k = 1 \dots K$ do
6          $\mathbf{H}^{(k)} = \text{ML-GNN}(\mathbf{H}^{(k-1)})$ // ML-GNN is a placeholder for either ML-GAT or ML-GCN
7      $\mathbf{Z} = \mathbf{H}^{(K)}$
8      Initialize empty entity-embedding matrix $\widetilde{\mathbf{Z}}$
       // Apply cross-layer aggregation to $Z$ as
9      for $l = 1 \dots \ell$ do
10         $\widetilde{\mathbf{Z}} = \widetilde{\mathbf{Z}} + \mu_l \mathbf{Z}_l$
11     Compute $\widehat{\mathbf{Z}} = ff(\widetilde{\mathbf{Z}})$, through a feed-forward neural network function $ff$
12     $\widehat{\mathbf{Z}}_{train} = \widehat{\mathbf{Z}}[\mathcal{V}_{train}]$ // Mask the unlabeled entities
13     $\widehat{\mathbf{Y}}_{train} = softmax(\widehat{\mathbf{Z}}_{train})$
14     Update all parameters w.r.t. the loss function applied to $\widehat{\mathbf{Y}}_{train}$ and $\mathbf{Y}_{train}$ (cf. Eq. 16)
15 $\widehat{\mathcal{Y}} = \arg\max softmax(\widehat{\mathbf{Z}})$ // Row-wise softmax and arg max over the second dimension of $\widehat{Z}$
```

In order to predict the class label for each entity $v \in \mathcal{V}$, we combine the node embeddings obtained from each layer. Given $\mathbf{Z}_l \in \mathbb{R}^{n_l\times d}$ as the learned embeddings for layer L_l, we obtain the entity representation through the following *cross-layer aggregation*:

$$\widetilde{\mathbf{Z}} = \sum_{l=1}^{\ell} \mu_l \mathbf{Z}_l, \tag{15}$$

where $\widetilde{\mathbf{Z}} \in \mathbb{R}^{N\times d}$ is the final entity embedding matrix, and $\mu \in \mathbb{R}^{\ell}$ is a vector of non-negative values that can be either pre-determined (e.g., uniform distribution, or any arbitrary user-provided distribution), or learned during the training through an optimization procedure. In our setting, we used the configuration with trainable parameters, as described in the following paragraph on parameter learning.

The last step is the application of a feed-forward neural network to the matrix $\widetilde{\mathbf{Z}}$ is provided in input to a feed-forward neural network, whose output is a matrix $\widehat{\mathbf{Z}} \in \mathbb{R}^{N\times C}$.

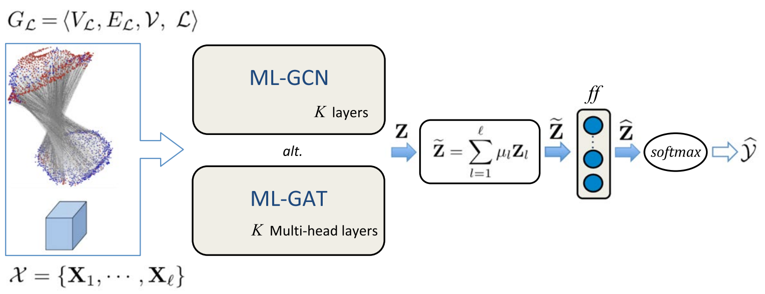

$G_{\mathcal{L}} = \langle V_{\mathcal{L}}, E_{\mathcal{L}}, \mathcal{V}, \mathcal{L} \rangle$
ML-GCN
K layers
alt.
ML-GAT
K Multi-head layers
$\mathcal{X} = \{\mathbf{X}_1, \cdots, \mathbf{X}_\ell\}$
$\mathbf{Z}$
$\widetilde{\mathbf{Z}} = \sum_{l=1}^{\ell} \mu_l \mathbf{Z}_l$
$\widetilde{\mathbf{Z}}$
ff
$\widehat{\mathbf{Z}}$
softmax
$\widehat{\mathcal{Y}}$

Fig. 2 Illustration of our proposed framework for node embedding generation and entity classification in a multilayer network

The final step consists in applying the row-wise softmax function onto $\widehat{\mathbf{Z}}$ to yield the class prediction of an entity. Algorithm 1 sketches the pseudo-code of our classification framework, whereas Fig. 2 depicts an illustration of the framework.

Parameter learning. With the exception of the cross-layer parameters μ, which are initialized by a uniform distribution (i.e., $\mu_l = \frac{1}{\ell}$, for all $l = 1 \ldots \ell$), all other parameters are initialized through Glorot (also known as Xavier) initialization (Glorot and Bengio 2010); this initialization technique is widely used in deep learning tasks and has proven to be effective in several applications (Mishkin and Matas 2016). Particularly, the trainable weight matrix $\mathbf{W}$ in ML-GAT and ML-GCN is subjected to Glorot initialization with normal distribution and with uniform distribution, respectively. Moreover, in ML-GAT, each layer of the multilayer network is also parametrized with a weight vector $\mathbf{Q} \in \mathbb{R}^{2d}$ of the single-layer feed forward neural network used as the attention mechanism (according to Velickovic et al. 2018).

All the above parameters, jointly with those of the neural network downstream of the cross-layer aggregation (cf. Eq. 15), are then updated during the training through an optimization strategy. That is, after the forward step of our learning framework, we calculate the loss function over all labeled examples $\mathcal{V}_{train}$. Then the gradient of the loss with respect to all parameters is calculated, which are finally updated through the optimization strategy (cf. "Experimental evaluation" section). We use the cross-entropy as supervised loss function shown in Eq. (16):

$$- \sum_{v \in \mathcal{V}_{train}} \sum_{c=1}^{C} Y_{v,c} \log \widehat{Y}_{v,c}. \tag{16}$$

Note that, like Kipf and Welling (2017), we avoid explicit graph-based regularization in the loss function, while our GNN models are learned through the supra-adjacency matrix of the multilayer network that allow the models to learn representation for nodes instances of unlabeled entities.

In order to show the outcomes of the representation learning process, a two-dimensional projection of the embeddings produced by the proposed approaches is reported in Fig. 3 (ML-GAT) and Fig. 4 (ML-GCN). The representation is obtained via the *t*-distributed stochastic neighbor embedding (*t-SNE*) method, which is a widely used nonlinear dimensionality reduction technique for embedding high-dimensional data for visualization in a low-dimensional space (van der Maaten and Hinton 2008). More specifically, the plots show the embeddings produced by t-SNE on $\widetilde{\mathbf{Z}}$ for the training entities (25% of total entities), i.e., the entity representation obtained through the cross-layer aggregation defined in Eq. (15). The embeddings refer to the *Koumbia*-2 network including its real-world node-attributes (cf. "Data" section); different colors in the figures correspond to the two different node labels. Left side of the figures shows the embeddings obtained after one training epoch of t-SNE, while the right one shows the final embeddings obtained after 1000 training epochs, extracted downstream of the second (i.e., last) hidden layer. It is easy to see how the embedding is already significant after only one training epoch, with a relatively good separation between the two classes (slightly more evident for ML-GAT). The embeddings get clearly better after 1000 training epochs, with an evident separation between the representations of entities belonging to the two classes.

Experimental evaluation

We evaluated our framework for semi-supervised entity classification tasks on several real-world multiplex and multilayer networks from different domains.

In the following we provide details on the evaluation network datasets, on the experimental settings of our proposed methods, and on that of competitors (GrAMME-SG and GrAMME-Fusion (Shanthamallu et al. 2020)) and baselines (GCN and GAT).

Data

We considered 9 network datasets, from which we derived a total of 19 networks for evaluation (plus their monoplex flattened versions). These datasets come from different domains and present very different structural characteristics. Moreover, all datasets are publicly available and most of them have been previously exploited as benchmarks for a variety of network analysis tasks, including node classification and link prediction. This is a major criterion for our choice, since it allows comparison of our results with the ones reported in previous literature. Eight of such datasets are originally provided as multiplex networks, i.e., networks in which inter-layer connections are coupling edges only, connecting a node and its counterparts in other layers. These networks also do not provide attributes associated to the entities. Therefore, in order to stress the ability of our framework of dealing with generic multilayer attributed networks, we introduced the *Koumbia* network dataset (Interdonato et al. 2020), which comes with unconstrained inter-layer connections and real-world properties associated to the entities. In the following we briefly describe our evaluation network datasets.

Balance (Siegler 1976) models psychological experimental results on a set of individuals. According to Shanthamallu et al. (2020), four attributes characterize the subjects (left weight, the left distance, the right weight, and the right distance). The classes correspond to the balance scale of the subjects (tip to the right, tip to the left, or being balanced).

CKM-Social (Coleman et al. 1957) contains social information from physicians (entities) in four towns in Illinois, Peoria, Bloomington, Quincy and Galesburg. It consists of 3 directed layers generated from different sociometric matrices, where the cities are used to label the entities.

Congress (Schlimmer 1987) models the results of bills obtained from the 1984 United States Congressional Voting Records Database. The network has 16 layers corresponding to votes, where for each layer two congressmen are linked if they voted the same. Each congressman is labeled as either democrat or republican.

DKPol (Magnani et al. 2021) (Dansk Politik) is a network with three types of online relations between Danish Members of the Parliament on Twitter. It comes with a ground truth corresponding to affiliations to 10 political parties, that we used as labels.

Leskovec-Ng (Chen and III 2016) is a 4-layer temporal collaboration network, which contains coauthors of Prof. Jure Leskovec or Prof. Andrew Ng over 20 years, partitioned into 4 different 5-year intervals. Entities are researchers, and on each layer there is an edge between two researchers if they co-authored at least one paper in the 5-year interval. Each researcher is labeled as Leskovec's collaborator or Ng's collaborator.

Starwars[2] is comprised of 6 layers, each corresponding to interactions between Starwars characters in the first 6 episodes of the saga. We manually labeled each character as male, female or droid and used this information as entity labels. Note that the resulting class distribution is very unbalanced as there are 76 males, 12 females and 4 droids.

Terrorist (Everton 2012) models interactions between 79 terrorists drawn from the Noordin's Network dataset. Similarly to Liu et al. (2017), we built a 4-layer network from the following relation types: trust, communication, operational, business and financial ties. We derive two different types of entity labels: (i) 2-class labels corresponding to membership of an individual to the Noordin's splinter group (member or non-member), and (ii) 3-class labels corresponding to the current state defined as the physical condition of the individual (dead, alive, jail). The two versions are dubbed **Terrorist-Noordin** and **Terrorist-status**, respectively.

Vickers (Vickers and Chan 1981) is a 3-layer directed multiplex network modeling the social relations between 29 seventh grade students in a school. We use the gender as entity labels; there are 12 boys and 17 girls.

Koumbia (Interdonato et al. 2020) is a multilayer network extracted from a Sentinel-2[3] satellite image time series, centered on an agricultural landscape in the Koumbia area in Burkina Faso. In this dataset, entities represent segments of the satellite image, and classes correspond either to crop (i.e., segments containing pixels related to cultivated area) or no-crop (i.e., segments containing pixels related to uncultivated areas, such as forests) segments. The network is originally associated with inter-layer edges and real-world attributes for the entities, corresponding to the segment statistics of the radiometric bands of the satellite images. We created the input feature matrix by concatenating the average values of ten different radiometric bands for 21 timestamps, obtaining a feature vector of size 210 for each entity. Fig. 5a displays the feature distribution obtained by linearizing the input feature matrix. Note that the *geo2net* framework[4] presented in Interdonato et al. (2020) is designed to build a multilayer network with an arbitrary number of layers, which model the association of nodes to an arbitrary number of functional classes (e.g., temporal radiometric profiles) by producing fuzzy layer memberships using

[2] https://github.com/evelinag/StarWars-social-network/tree/master/networks.

[3] https://sentinel.esa.int/web/sentinel/missions/sentinel-2.

[4] https://gitlab.irstea.fr/raffaele.gaetano/geo2net.

the *fuzzy c-means* algorithm (Ross 2009). In our case study, we exploit this functionality in order to take into account versions of the network with a varying number of layers (i.e., 2, 5, 10, 15, 20). We denote each of these networks as *Koumbia-l* (e.g., *Koumbia*-5 will denote the version with 5 layers). Figure 5 shows *Koumbia*-2 (b) and *Koumbia*-5 (c) graphs, whereas Table 4 reports detailed information about inter-layer edges. Since the competitors are specifically designed for multiplex networks, for a fair comparison with those methods we will also use a multiplex version of this dataset (i.e., obtained discarding inter-layer connections other than coupling ones), named *Koumbia-l-mpx*.

Table 2 summarizes the structural properties of the all network datasets, plus the multiplex versions of *Koumbia*. For the latter, inter-layer edge information is reported in Table 4. Note that the graph homophily in Table 2 corresponds to the fraction of edges in a graph connecting nodes with the same class label (Zhu et al. 2020), formally $|\{(v_i, v_j) : (v_i, v_j) \in E \wedge y_i = y_j\}|/|E|$, with $v_i, v_j \in V$ and y_i, y_j class labels of v_i, v_j, respectively. This statistic was calculated excluding the inter-layer edges, which otherwise would lead to biased homophily scores. Also, average entity frequency corresponds to the fraction of layers on which each entity appears, averaged over all entities. Moreover, Table 3 summarizes information on the monoplex, flattened versions of the network datasets, that will be used for evaluation of baseline methods designed for monoplex networks.

Experimental settings

We conducted all experiments under a transductive learning setting whereby, given an input multilayer network, only a fixed portion of the set of entities for each class were used as labeled data for the training of a GNN model. Recall that, due to the transductive setup, the learning process is nonetheless able to use all node attributes and topological information. For those networks without external information, node attributes were initialized by sampling each attribute from either a Gaussian, an Exponential, or a Uniform distributions; more precisely, for a given choice of number f of node attributes, either we randomly generated each attribute values from a Gaussian distribution, or we randomly generated one third each of the attributes from Gaussian, Exponential and Uniform distributions.

We used two main settings for the training set size, namely at 25% and 5% of the set of entities (we will refer to the setting with 25% of training set size unless otherwise specified). All GNNs were trained using the Adam optimization algorithm (Kingma and Ba 2017) with full batch size, for either 1000 epochs, or at convergence when the early-stopping regularization technique was used (with patience value of 50 epochs), learning rate set to 0.005, $L2$ weight regularization set to 0.0005, and dropout regularization technique with $p = 0.6$ applied to the hidden layers and to the normalized attention coefficients of GCN and GAT based methods, respectively. Note that this introduces stochasticity since we sample the within-layer and outside-layer neighborhood of each node. It should also be noted that, since all our evaluation networks fit into the GPU memory, we performed full batch training (Kipf and Welling 2017, 2016), where the parameters are updated after processing the whole network. Also, regarding the cross-layer aggregation (cf. Eq. 15), the parameters μ were initialized with uniform distribution. Both our ML-GAT method

Table 2 Summary of structural characteristics of the network datasets: type of the network, number of layers (l), number of entities, or actors (a), number of nodes (n), number of edges (e), density mean/SD over the layers, node degree mean/SD over the layers, average entity frequency (aef), homophily, and number of classes (c)

Network	Type	l	a	n	e	density	degree	aef (%)	homophily	c
Balance	Social	4	625	2500	155,000	0.199 ± 0.0	124.0 ± 0.0	100	0.507 ± 0.0	3
CKM-Social	Social	3	241	723	2740	0.016 ± 0.001	7.580 ± 0.632	100	1.0 ± 0.0	4
Congress	Vote	16	435	6960	707,872	0.496 ± 0.025	203.411 ± 10.86	100	0.670 ± 0.115	2
DKPol	Political	3	490	839	20,226	0.07 ± 0.008	28.991 ± 44.115	57.07	0.518 ± 0.282	10
Leskovec-Ng	Collaboration, Temporal	4	191	214	536	0.166 ± 0.151	4.088 ± 1.454	28.01	0.994 ± 0.012	2
Starwars	Interaction	6	92	157	494	0.253 ± 0.045	6.117 ± 0.835	28.44	0.571 ± 0.067	3
Terrorist-Noordin	Terrorist	4	79	227	1822	0.129 ± 0.05	13.829 ± 9.087	71.83	0.780 ± 0.118	2
Terrorist-status	Terrorist	4	79	227	1822	0.129 ± 0.05	13.829 ± 9.087	71.83	0.469 ± 0.069	3
Vickers	Social	3	29	87	740	0.304 ± 0.122	17.011 ± 6.854	100	0.763 ± 0.088	2
Koumbia-2-mpx	Geospatial	2	2246	2246	6011	0.002 ± 0.001	5.355 ± 0.017	50	0.814 ± 0.004	2
Koumbia-5-mpx	Geospatial	5	2246	3776	12,870	0.005 ± 0.002	6.751 ± 0.448	33.62	0.827 ± 0.097	2
Koumbia-10-mpx	Geospatial	10	2246	7191	27,433	0.006 ± 0.003	7.567 ± 0.392	32.02	0.826 ± 0.091	2
Koumbia-15-mpx	Geospatial	15	2246	10,591	43,285	0.007 ± 0.003	8.103 ± 0.458	31.43	0.826 ± 0.090	2
Koumbia-20-mpx	Geospatial	20	2246	14,069	59,968	0.008 ± 0.07	8.447 ± 0.489	31.32	0.823 ± 0.089	2

Table 3 Summary of structural characteristics of the flattened (monoplex) versions of the evaluation networks: number of nodes (n), number of edges (e), average density (den), and average node degree (deg)

Network	n	e	den	deg
Balance	625	115,000	0.590	368.000
CKM-Social	241	1848	0.032	15.336
Congress	435	93,327	0.989	429.090
DKPol	490	19,638	0.164	80.155
Leskovec-Ng	191	511	0.028	5.351
Starwars	92	380	0.093	8.352
Terrorist	79	1246	0.207	31.949
Vickers	29	376	0.463	25.931
Koumbia-2	2246	6011	0.001	5.353
Koumbia-5	2246	11,404	0.002	10.155
Koumbia-10	2246	17,545	0.003	15.623
Koumbia-15	2246	22,533	0.004	20.065
Koumbia-20	2246	25,748	0.005	22.928

and GAT use multi-head attention with $Q = 2$ attentions heads for each layer where the heads are averaged in order to save memory resource. Furthermore, the negative slope β for LeakyReLU function was set to 0.2 (cf. "Background" section). We set $K = 2$ with $d = 32$ features, and number of input features to $f = 64$.

Concerning the setting of baseline methods, the original GAT and GCN methods were trained over the flattened networks (cf. Table 3), since they were conceived for single-layer graphs. Moreover, since the GrAMME (Shanthamallu et al. 2020) framework can deal with multirelational/multiplex networks only, in our comparative experiments we used the multiplex versions of the *Koumbia-l* networks, named *Koumbia_mpx*, thus discarding inter-layer edges connecting nodes representing different entities.

For GrAMME-SG and GrAMME-Fusion, we set learning rate to 0.01, $f = 64$, $d = 32$, 5 fusion heads for GrAMME-Fusion, and used dropout regularization technique.

Note that we used the publicly available software implementations for all the competitors.[5]

Results

We present the results of the experimental evaluation of the proposed framework, which are organized into two evaluation stages:

1. In "Evaluation with competing methods" section, we compare the proposed methods and competitors (i.e., GrAMME-SG, GrAMME-Fusion, GAT and GCN), on the multiplex networks introduced in "Data" section, in terms of prediction performance, impact due to the setting of the training set size and early-stopping technique, and

[5] The GrAMME, GAT and GCN source codes are publicly available at https://github.com/udayshankars/, https://github.com/Diego999/pyGAT, and https://github.com/tkipf/pygcn, respectively.

Table 4 Inter-layer edge information of *Koumbia* multilayer networks for different number of layers (Interdonato et al. 2020): number of inter-layer edges (e), inter-layer average degree (deg)

ℓ	e	deg
2	3788	3.37
5	22,918	20.41
10	99,084	88.23
15	225,540	200.84
20	406,254	361.76

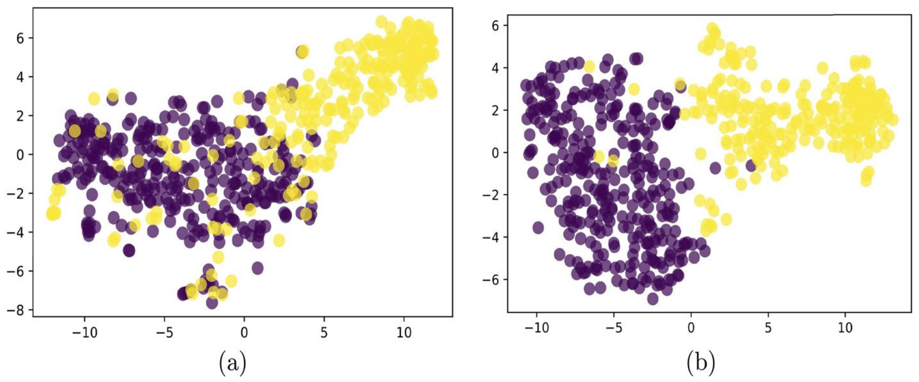


Fig. 3 *t*-SNE 2D visualization of the entity embeddings learned from ML-GAT over *Koumbia*-2: **a** after one epoch, **b** after 1000 epochs

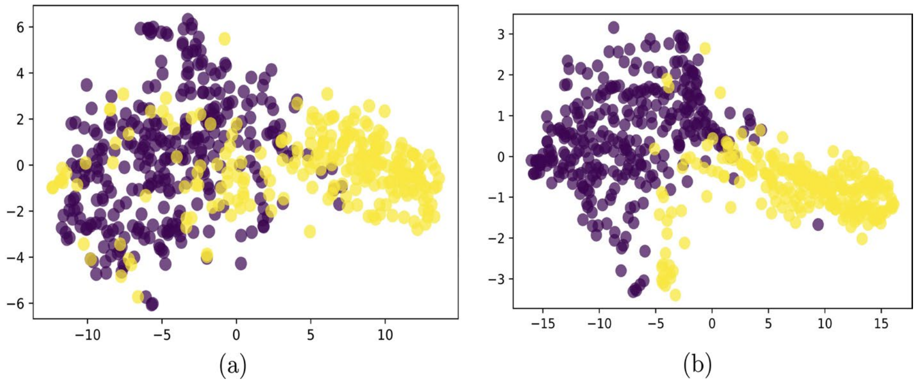


Fig. 4 *t*-SNE 2D visualization of the entity embeddings learned from ML-GCN over *Koumbia*-2: **a** after one epoch, **b** after 1000 epochs

of the initial input node-features. We also analyze the training execution times of the methods, and we discuss aspects of their computational complexity.

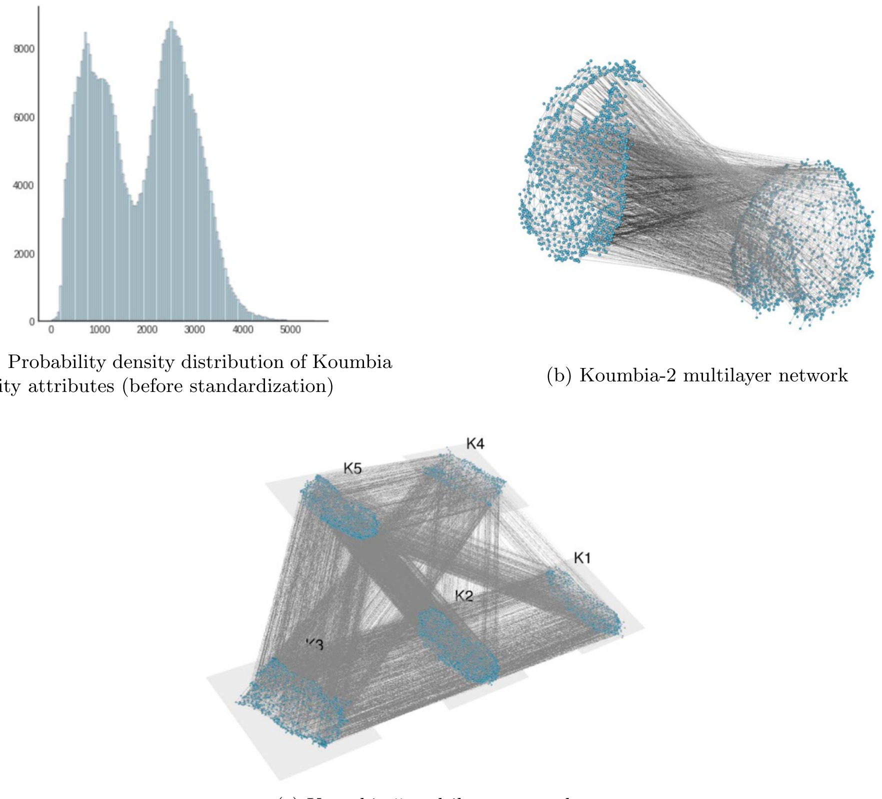
2. In "Evaluation on real-world node-attributes and arbitrary inter-layer edges: the Koumbia multilayer network testbed" section, we conduct a thorough analysis of the Koumbia dataset, focusing on the impact of using real node-features. We focus on this network since it is the only one including real-world attributes for the entities

and arbitrary inter-layer edges, thus allowing to evaluate to what extent the proposed framework is able to exploit such characteristics.

Evaluation with competing methods

Table 5 reports the average accuracy scores achieved by the proposed ML-GAT and ML-GCN approaches, and the competing methods (GrAMME-SG, GrAMME-Fusion, GAT and GCN). For each network and method, the average accuracy was computed over 10 independent runs, where each run corresponded to a different train-test split, with 25% of training entities, and the input features of the entities were randomly initialized with a normal distribution for all the networks.

At a first glance, our proposed methods are able to achieve very high, or even nearly optimal accuracy, on most networks. This also mostly holds for GrAMME-Fusion and, to a less extent, for GrAMME-SG. Moreover, the results obtained by the baselines on the corresponding monoplex versions of the network datasets reveal some situations in which the flattening process is actually beneficial for the entity classification task: particularly, as it can be observed in Fig. 6, *CKM-social* and *Leskovec-Ng* layers are mostly structured with disconnected components, where each component in a layer contains nodes belonging to a specific label, so that the monoplex flattened network can better



(a) Probability density distribution of Koumbia entity attributes (before standardization)

(b) Koumbia-2 multilayer network

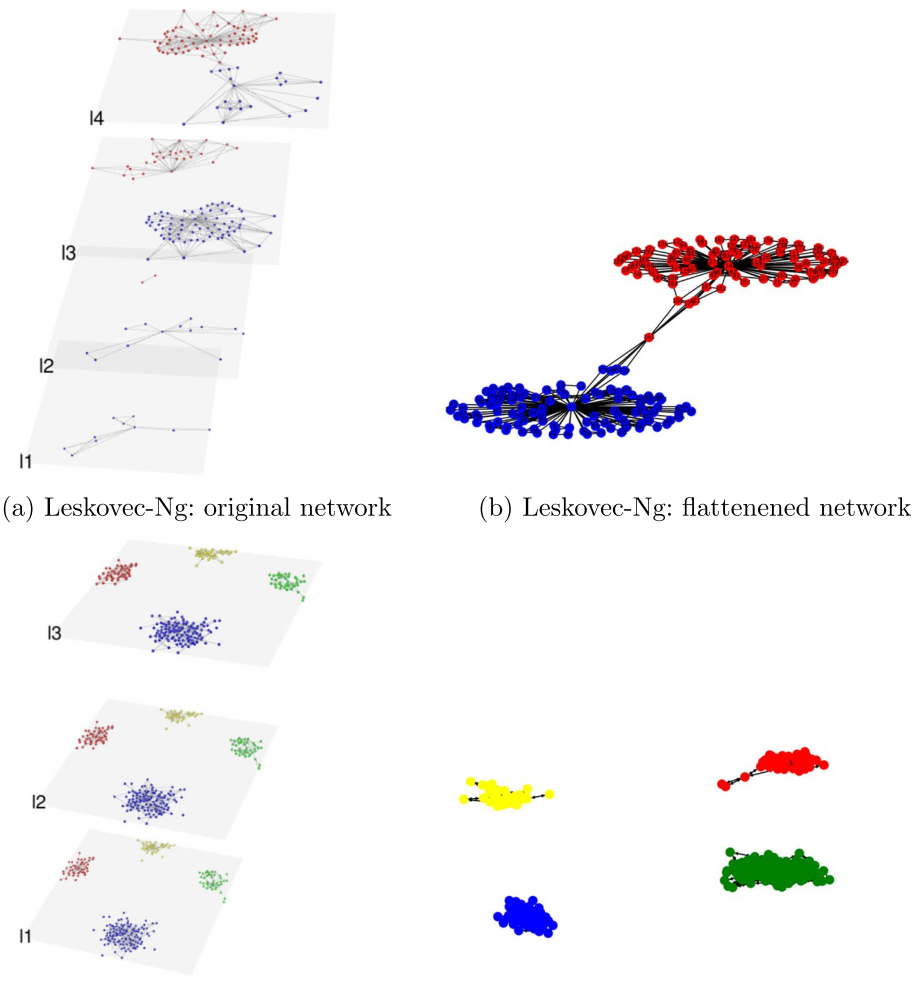
(c) Koumbia-5 multilayer network

Fig. 5 Feature distribution of Koumbia entity attributes (obtained by linearizing the input feature matrix) and visual representation of Koumbia-2 and Koumbia-5 with inter-layer and intra-layer edges

Table 5 Accuracy (mean and standard deviation over 10 runs) obtained by the proposed methods and competitors

Network	ML-GAT	ML-GCN	GrAMME-SG	GrAMME-Fusion	GAT	GCN
Balance	0.8582 ± 0.0283	**0.9306 ± 0.0216**	0.8004 ± 0.0244	0.8230 ± 0.0764	0.6445 ± 0.0720	0.8101 ± 0.0533
CKM-Social	0.9544 ± 0.0113	0.8242 ± 0.0420	0.9685 ± 0.0138	0.9807 ± 0.0075	**0.9962 ± 0.0052**	0.9736 ± 0.0121
Congress	0.9236 ± 0.0275	0.9362 ± 0.0217	0.8834 ± 0.0000	**0.9601 ± 0.0000**	0.5370 ± 0.0953	0.6146 ± 0.0000
DKPol	0.8332 ± 0.0163	0.7866 ± 0.0237	0.5590 ± 0.0256	0.8424 ± 0.0212	0.8535 ± 0.0206	**0.8838 ± 0.0151**
Leskovec-Ng	**0.9959 ± 0.0036**	0.9243 ± 0.0405	0.9545 ± 0.0422	0.9797 ± 0.0022	0.9903 ± 0.0128	0.9937 ± 0.0022
Starwars	0.7000 ± 0.0788	0.7140 ± 0.0477	0.6319 ± 0.0771	**0.7754 ± 0.0370**	0.7464 ± 0.0300	0.6855 ± 0.0611
Terrorist-Noordin	0.7150 ± 0.0669	0.7833 ± 0.0801	0.7000 ± 0.0939	**0.8237 ± 0.0322**	0.7133 ± 0.0706	0.6667 ± 0.0720
Terrorist-status	0.4767 ± 0.0439	0.5022 ± 0.0835	0.4915 ± 0.0554	0.4932 ± 0.0847	0.4650 ± 0.0748	**0.5034 ± 0.0809**
Vickers	**0.9692 ± 0.0372**	0.9591 ± 0.0725	0.8227 ± 0.1311	0.8909 ± 0.1563	0.9136 ± 0.0693	0.8227 ± 0.0756
Koumbia-2-mpx	**0.7937 ± 0.0106**	0.7578 ± 0.0434	0.6663 ± 0.0170	0.7520 ± 0.0064	0.7485 ± 0.0231	0.7031 ± 0.0107
Koumbia-5-mpx	**0.8404 ± 0.0132**	0.8314 ± 0.0289	0.6425 ± 0.0000	0.8040 ± 0.0000	0.8077 ± 0.0128	0.7802 ± 0.0167
Koumbia-10-mpx	**0.8425 ± 0.0093**	0.8192 ± 0.0011	na	na	0.8420 ± 0.0058	0.8147 ± 0.0094

Bold values refer to the best results on each network



(a) Leskovec-Ng: original network (b) Leskovec-Ng: flattened network

(c) CKM-Social: original network (d) CKM-Social: flattened network

Fig. 6 *Leskovec-Ng* and *CKM-Social* networks before and after the flattening process. In *Leskovec-Ng*, blue nodes and red nodes correspond to Leskovec's collaborators and to Ng's collaborators, respectively. In *CKM-Social*, color codes correspond to different cities

contextualize each node with respect to neighbors of different classes. Note also how the attention-based approaches (i.e., ML-GAT and GrAMME) do not suffer from the same issue, and obtain performances close to or even better than the ones of the baselines.

By contrast, multilayer approaches always obtain the best performances on networks with very high values of average entity frequency (i.e. the average percentage of layers on which each entity appears, cf. Table 2), such as *Vickers* (ML-GAT), *Congress* (GrAMME-Fusion) and *Balance* (ML-GCN); an exception is represented by *CKM-Social*, which is explained since in this network the nodes in every layer are associated with the same label. The latter case is also interesting as it is the only one where ML-GCN outperforms not only the competitors but also ML-GAT (which is nonetheless the second best performer). This might be ascribed to the peculiar structure of the *Balance* network, where each layer is composed by 5 disjoint complete connected-components and the various node labels are distributed over the components. In fact, ML-GAT has worse

Table 6 Accuracy (mean and standard deviation over 10 runs) obtained by the proposed methods and competitors

Network	ML-GAT	ML-GCN	GrAMME-SG	GrAMME-Fusion	GAT	GCN
Balance	0.7265 ± 0.0270	**0.8164 ± 0.0139**	0.6795 ± 0.0377	0.7188 ± 0.0483	0.5496 ± 0.0852	0.7670 ± 0.0317
CKM-Social	0.8652 ± 0.0193	0.7819 ± 0.0213	0.8026 ± 0.0649	0.8638 ± 0.0596	**0.9815 ± 0.0189**	0.9427 ± 0.0281
Congress	0.9262 ± 0.0212	**0.9342 ± 0.0065**	0.8814 ± 0.0000	0.8879 ± 0.0000	0.5398 ± 0.0649	0.6146 ± 0.0000
DKPol	0.6809 ± 0.0510	0.5377 ± 0.2600	0.3679 ± 0.0377	0.5373 ± 0.0513	0.7384 ± 0.0577	**0.7969 ± 0.0394**
Leskovec-Ng	**0.9872 ± 0.0053**	0.8750 ± 0.0454	0.7873 ± 0.0960	0.8746 ± 0.0749	0.9867 ± 0.0191	0.9867 ± 0.0146
Starwars	0.7465 ± 0.0455	**0.7767 ± 0.0741**	0.6885 ± 0.1278	0.6828 ± 0.1861	0.6023 ± 0.0757	0.6151 ± 0.1531
Terrorist-Noordin	0.6568 ± 0.0428	0.7284 ± 0.0149	0.6536 ± 0.0947	**0.7427 ± 0.0860**	0.6649 ± 0.0774	0.6324 ± 0.0734
Terrorist-status	0.4648 ± 0.0377	0.4767 ± 0.0328	**0.5460 ± 0.0790**	0.5360 ± 0.0917	0.4575 ± 0.0810	0.4247 ± 0.0961
Vickers	**0.9296 ± 0.0729**	0.9074 ± 0.0400	0.5750 ± 0.1745	0.5643 ± 0.1355	0.7296 ± 0.1063	0.7851 ± 0.0716
Koumbia-2-mpx	**0.6551 ± 0.0121**	0.6405 ± 0.0448	0.5999 ± 0.0188	0.6404 ± 0.0064	0.5865 ± 0.0182	0.5835 ± 0.0187
Koumbia-5-mpx	**0.7457 ± 0.0286**	0.6677 ± 0.0194	0.5455 ± 0.0156	0.7147 ± 0.0120	0.6698 ± 0.0162	0.6426 ± 0.0339
Koumbia-10-mpx	**0.7674 ± 0.0596**	0.6265 ± 0.0252	na	na	0.7278 ± 0.0179	0.6694 ± 0.0156

Training and testing set sizes correspond to 5% and 95% of the entities, respectively

Bold values refer to the best results on each network

Table 7 Accuracy (mean and standard deviation over 10 runs) obtained by the proposed methods and competitors, with early-stopping regularization technique

Network	ML-GAT	ML-GCN	GrAMME-SG	GrAMME-Fusion	GAT	GCN
Balance	0.8885 ± 0.0208	**0.8965 ± 0.0990**	0.7543 ± 0.0426	0.7982 ± 0.0666	0.7244 ± 0.0910	0.7881 ± 0.0270
CKM-Social	0.9623 ± 0.0088	0.9213 ± 0.0226	0.9265 ± 0.0253	0.9479 ± 0.0192	**0.9762 ± 0.0179**	0.9729 ± 0.0173
Congress	0.9431 ± 0.0195	**0.9440 ± 0.0089**	0.8623 ± 0.0231	0.9232 ± 0.0303	0.6116 ± 0.0092	0.6146 ± 0.0000
DKPol	0.8294 ± 0.0298	0.7681 ± 0.0122	0.5910 ± 0.0433	0.8233 ± 0.0277	0.7960 ± 0.0294	**0.8621 ± 0.0321**
Leskovec-Ng	**1.0 ± 0.0000**	0.9570 ± 0.0178	0.9516 ± 0.0242	0.9832 ± 0.0120	0.9792 ± 0.0202	0.9861 ± 0.0155
Starwars	**0.8174 ± 0.0183**	0.8152 ± 0.0185	0.6348 ± 0.1001	0.6522 ± 0.1108	0.8087 ± 0.0171	0.7869 ± 0.0225
Terrorist-Noordin	0.6500 ± 0.0745	0.7325 ± 0.0566	0.7333 ± 0.1141	**0.7179 ± 0.1216**	0.7175 ± 0.0426	0.6850 ± 0.0818
Terrorist-status	0.5700 ± 0.0483	0.5450 ± 0.0369	0.5333 ± 0.0380	0.4974 ± 0.0429	**0.6313 ± 0.0944**	0.5278 ± 0.0870
Vickers	**0.9733 ± 0.0466**	0.9667 ± 0.0471	0.8769 ± 0.1032	0.8974 ± 0.0544	0.8667 ± 0.0544	0.8000 ± 0.0943
Koumbia-2-mpx	**0.7890 ± 0.0134**	0.7671 ± 0.0173	0.6403 ± 0.0815	0.7333 ± 0.0020	0.6058 ± 0.0138	0.6846 ± 0.0139
Koumbia-5-mpx	**0.8431 ± 0.0127**	0.7995 ± 0.0114	0.5481 ± 0.0013	0.8021 ± 0.0150	0.7373 ± 0.0471	0.7715 ± 0.0141
Koumbia-10-mpx	**0.8419 ± 0.0105**	0.8151 ± 0.0162	na	na	0.7825 ± 0.0612	0.8087 ± 0.0161

Training, validation and testing set sizes correspond to 25%, 25% and 50% of the entities, respectively

Bold values refer to the best results on each network

Table 8 Accuracy (mean and standard deviation over 10 runs) obtained by the proposed methods and competitors, with early-stopping regularization technique

Network	ML-GAT	ML-GCN	GrAMME-SG	GrAMME-Fusion	GAT	GCN
Balance	0.7642 ± 0.0416	**0.7759 ± 0.0244**	0.5748 ± 0.1137	0.6555 ± 0.0711	0.6046 ± 0.1117	0.7282 ± 0.0451
CKM-Social	0.8771 ± 0.0228	0.7882 ± 0.0424	0.7293 ± 0.0779	0.8882 ± 0.0250	**0.9512 ± 0.0307**	0.9288 ± 0.0331
Congress	0.9111 ± 0.0271	**0.9370 ± 0.0114**	0.8683 ± 0.0210	0.8581 ± 0.0107	0.6131 ± 0.0000	0.6131 ± 0.0000
DKPol	0.6234 ± 0.0566	0.6055 ± 0.0343	0.3345 ± 0.0587	0.5392 ± 0.1015	0.5992 ± 0.1034	**0.7789 ± 0.0421**
Leskovec-Ng	**0.9828 ± 0.0145**	0.8918 ± 0.0376	0.6673 ± 0.1151	0.8110 ± 0.0861	0.9724 ± 0.0190	0.9746 ± 0.0223
Starwars	0.7894 ± 0.0353	**0.8121 ± 0.0106**	0.6156 ± 0.0109	0.6656 ± 0.0650	0.7500 ± 0.0464	0.7318 ± 0.0982
Terrorist-Noordin	0.7036 ± 0.0339	**0.7214 ± 0.0674**	0.6473 ± 0.1009	0.6618 ± 0.0419	0.6625 ± 0.0586	0.6893 ± 0.0620
Terrorist-status	0.5070 ± 0.0791	0.5554 ± 0.0638	0.4873 ± 0.1131	0.4764 ± 0.0505	**0.5643 ± 0.0192**	0.4921 ± 0.1202
Vickers	**0.9571 ± 0.0417**	0.9143 ± 0.0438	0.6842 ± 0.754	0.7211 ± 0.1907	0.6428 ± 0.1510	0.6714 ± 0.1132
Koumbia-2-mpx	**0.6540 ± 0.0258**	0.6376 ± 0.0250	0.5641 ± 0.0211	0.6306 ± 0.1085	0.5662 ± 0.0394	0.5784 ± 0.0110
Koumbia-5-mpx	**0.7585 ± 0.0501**	0.6715 ± 0.0177	0.5525 ± 0.0088	0.7480 ± 0.0401	0.6051 ± 0.0433	0.6331 ± 0.0234
Koumbia-10-mpx	**0.8080 ± 0.0445**	0.7019 ± 0.0493	na	na	0.6613 ± 0.0620	0.6681 ± 0.0305

Training, validation and testing set sizes correspond to 5%, 25% and 70% of the entities, respectively

Bold values refer to the best results on each network

Table 9 Accuracy (mean and standard deviation over 10 runs) obtained by initializing the entity features with mixed distributions

Network	ML-GAT	ML-GCN	GAT	GCN
Balance	0.8377 ± 0.0567	**0.9053 ± 0.0168**	0.6362 ± 0.0346	0.6714 ± 0.0489
CKM-Social	0.9473 ± 0.0231	0.8066 ± 0.0399	**0.9962 ± 0.0069**	0.9786 ± 0.0075
Congress	0.8900 ± 0.1294	**0.9468 ± 0.0165**	0.5458 ± 0.1108	0.6146 ± 0.0000
DKPol	0.7935 ± 0.0245	**0.8205 ± 0.0323**	0.7956 ± 0.0646	0.7919 ± 0.0424
Leskovec-Ng	0.9944 ± 0.0085	0.9208 ± 0.0297	**0.9951 ± 0.0087**	0.9861 ± 0.0131
Starwars	0.7435 ± 0.0438	0.7217 ± 0.0625	**0.7782 ± 0.0493**	0.7202 ± 0.0552
Terrorist-Noordin	0.7133 ± 0.0680	**0.7550 ± 0.0643**	0.7017 ± 0.0337	0.6983 ± 0.0569
Terrorist-status	0.4883 ± 0.0629	0.4700 ± 0.1018	**0.5417 ± 0.0568**	0.5183 ± 0.0678
Vickers	**0.9773 ± 0.0321**	0.9636 ± 0.0469	0.7727 ± 0.1071	0.8590 ± 0.0452
Koumbia-2-mpx	0.7162 ± 0.0395	**0.7634 ± 0.0092**	0.7006 ± 0.0171	0.7020 ± 0.0200
Koumbia-5-mpx	**0.8497 ± 0.0092**	0.8046 ± 0.0212	0.7759 ± 0.0301	0.7702 ± 0.0143
Koumbia-10-mpx	**0.8579 ± 0.0148**	0.8228 ± 0.0159	0.8361 ± 0.0123	0.8052 ± 0.0110
Koumbia-15-mpx	**0.8551 ± 0.0075**	0.8223 ± 0.0105	0.8539 ± 0.0053	0.8181 ± 0.0121
Koumbia-20-mpx	**0.8561 ± 0.0157**	0.8305 ± 0.0053	0.8549 ± 0.0155	0.8312 ± 0.0125

Bold values refer to the best results on each network

performance than ML-GCN for both the *Balance* and *Congress* networks, which are the two with highest combination of average degree and average density (cf. Table 2). As already observed in Mohan (2021), for networks with high average degree and dense supra-adjacency matrix, GCN-based methods may perform better than random-walk based approaches and GAT based ones due to the stochasticity introduced by the attention mechanism. Furthermore, according to Zhu et al. (2020) and Qian et al. (2021), all models have relatively low performance on the network with the lowest homophily score, i.e., *Terrorist-status* (0.469). Likewise, all models have good performance on networks with strong homophily, such as *Leskovec-Ng* (0.994) and *CKM-Social* (1.0). On the other hand, note how multilayer models can still perform well on networks with lower homophily.

Finally, a major remark that stands out from Table 5 concerns the results obtained on *Koumbia* networks, for increasing number of layers. Note that *Koumbia* is the network dataset including the highest number of entities (2246), and that the *Koumbia*-2 network has no edge overlap between the two layers (i.e., the node set on the two layers is completely disjoint). Notably, our ML-GAT followed by ML-GCN outperform all the competitors, with GrAMME-SG performing even worse than baselines GAT and GCN. Note also that GrAMME revealed to be sensitive to the number of layers, at the point that for 10 layers (i.e., *Koumbia*-10-*mpx*) both variants of our major competitor run out-of-running-time.

Impact of training set size and early-stopping

We investigated a more challenging scenario for the training of the GNN models under study, by using only 5% of the entities as training instances, and the remaining ones for testing. Results are reported in Table 6. As expected, the classification performances of the various methods tend to decrease in almost all the networks. Two particular situations occur with the *Starwars* and *Terrorist-status* networks: on the former, only

Table 10 Comparison of training time (minutes) between ML-GAT, ML-GCN, GrAMME-SG and GrAMME-Fusion

Network	ML-GAT	ML-GCN	GrAMME-SG	GrAMME-Fusion
Balance	0.2935	0.1270	12.9228	12.7558
CKM-Social	0.2465	0.1275	3.0005	2.6893
DKPol	0.2626	0.1798	7.2329	6.2134
Leskovec-Ng	0.2632	0.1269	3.1790	2.4235
Vickers	0.2663	0.1255	0.3158	0.3242
Koumbia-2-mpx	0.2705	0.1727	52.3738	44.1908

The training was performed on Google Colab with Tesla T4 GPU with the same hyperparameter setting described in "Experimental settings" section

GrAMME-Fusion and GCN have worse performance, while on the latter, GrAMME methods even improve w.r.t. Table 5. These can be explained as both network datasets have highly unbalanced distribution of class labels, therefore for the least covered class (i.e., 'droid' for *Starwars* and 'dead' for *Terrorist-status*) the number of selected training instances does not significantly change as the percentage of training set size decreases from 25 to 5%. More interestingly, on the largest networks other than *Koumbia*, i.e., *Congress* and *Balance*, ML-GCN and ML-GAT outperform the other competing models. In general, it turns out to be that ML-GCN and ML-GAT tend to be less sensitive than the other methods when the percentage of training set size changes from 25% to 5%.

Another important aspect that we have not considered so far is the opportunity of using the early-stopping regularization which, with the use of a validation set, can be helpful to mitigate over-fitting of the GNN models. To this purpose, we carried out a further stage of evaluation where each GNN model was equipped with early-stopping and a patience value of 50 epochs, i.e., the training of a GNN model was terminated if the validation accuracy had not increased for 50 consecutive epochs. Tables 7 and 8 show results corresponding to 25% and 5% of training set size, respectively. Considering first the effect of early-stopping with training set size of 25%, we observe that our ML-GAT and ML-GCN and their monoplex counterparts improve their performance w.r.t. the scenario without early-stopping (i.e., Table 5) in most of the networks. For instance, ML-GAT and ML-GCN increase their accuracy on *Starwars*, from 0.70 to 0.817 and from 0.714 to 0.815, on *Terrorist-status*, from 0.477 to 0.570 and from 0.502 to 0.545, on *CKM-Social*, from 0.954 to 0.962 and from 0.824 to 0.921, respectively. Moreover, on *Koumbia* networks, ML-GAT and ML-GCN achieve comparable or even better results (i.e., ML-GAT on *Koumbia*-5-*mpx*) than those corresponding to non-early-stopping, while the monoplex counterparts, especially GAT, decrease their performance significantly. By contrast, GrAMME methods tend to benefit less from the use of early-stopping.

Finally, from the comparison between Tables 6 and 8 corresponding to 5% of training set size, we draw analogous remarks to the above discussed for the scenario with 25% of training set size. Although the variations between results in Table 8 and corresponding results in Table 6 are in general relatively small, some cases are still remarkable, such as the improvement of ML-GAT and ML-GCN on the two *Terrorist* networks, *Koumbia*-5-*mpx* and *Koumbia*-10-*mpx*.

Table 11 Performance of ML-GAT over the *Koumbia* multilayer networks with varying number of layers, and different types of input features

ℓ	*Koumbia-Fon*		*Koumbia-Foff*		*Koumbia-normal*	
	Accuracy	**MRR**	**Accuracy**	**MRR**	**Accuracy**	**MRR**
2	**0.9082** $\pm$ **0.0113**	**0.9541** $\pm$ **0.0057**	0.7426 $\pm$ 0.0200	0.8713 $\pm$ 0.0100	0.7556 $\pm$ 0.0164	0.8778 $\pm$ 0.0082
5	**0.9373** $\pm$ **0.0054**	**0.9687** $\pm$ **0.0027**	0.7656 $\pm$ 0.0117	0.8828 $\pm$ 0.0059	0.8355 $\pm$ 0.01153	0.9193 $\pm$ 0.0076
10	**0.9385** $\pm$ **0.0073**	**0.9693** $\pm$ **0.0036**	0.7588 $\pm$ 0.0126	0.8794 $\pm$ 0.0063	0.7874 $\pm$ 0.0116	0.8937 $\pm$ 0.0058
15	**0.9355** $\pm$ **0.0089**	**0.9687** $\pm$ **0.0044**	0.7522 $\pm$ 0.0125	0.8761 $\pm$ 0.0063	0.7723 $\pm$ 0.0133	0.8826 $\pm$ 0.0067
20	**0.9404** $\pm$ **0.0069**	**0.9702** $\pm$ **0.0035**	0.7488 $\pm$ 0.0167	0.8744 $\pm$ 0.0084	0.7706 $\pm$ 0.0091	0.8853 $\pm$ 0.0046

Bold values correspond to best performances

Impact of the attribute matrix

To better assess the robustness of our proposed methods, and also to gain insights into those cases in favor of monoplex-based baselines, we replicated the previous analysis by initializing the entity features with mixed distributions, following the other, more realistic approach described in "Experimental settings" section, i.e., one third of the attributes are modeled as normal distributions, one third as uniform distributions, and one third as exponential distribution.

Results of these experiments are reported in Table 9. While GAT and GCN are still the best performing methods on *Leskovec-Ng* and *CKM-Social*—due to the peculiar structural characteristics described in the above section, that makes the monoplex version better suited for the task at hand than the multilayer network—it can be noted that their relative performances on *DKPol* are significantly decreased (about 0.79) w.r.t. the ones observed in Table 5; by contrast, on this network, ML-GCN is the best performing method (0.82). Also, the accuracy of GAT (0.77) and GCN (0.67) worsens significantly for *Vickers* and *Balance*, respectively. By contrast, ML-GCN achieves better accuracy than the corresponding values in Table 5 on *DKPol*, *Congress*, and *Koumbia-2-mpx*, while both ML-GAT and ML-GCN improve on *Vickers* and *Starwars*.

Overall, comparing the methods' accuracy values averaged over the networks, from Table 9 w.r.t. Table 5, GAT and GCN show a more evident decrease percentage (resp., about −2% and −1.5%) than ML-GAT and ML-GCN. To sum up, while the performances of the monoplex-based baselines change drastically in some cases, indicating more sensitivity to the characteristics of the node attributes, our proposed methods turn out to be more robust, even benefiting from a mixed, thus more realistic, distribution of values for the node attributes in some networks.

Computational complexity aspects and training time analysis

In this section, we first discuss the computational complexity of our methods, ML-GAT and ML-GCN, assuming sparse supra-adjacency matrix and that the total number of nodes in a multilayer network is $\mathcal{O}(N\ell)$.

Table 12 Performance of ML-GCN over the *Koumbia* multilayer networks with varying number of layers, and different types of input features

ℓ	*Koumbia-Fon*		*Koumbia-Foff*		*Koumbia-normal*	
	Accuracy	**MRR**	**Accuracy**	**MRR**	**Accuracy**	**MRR**
2	**0.8520 ± 0.0863**	**0.9260 ± 0.0431**	0.7444 ± 0.0172	0.8722 ± 0.0086	0.7503 ± 0.0186	0.8751 ± 0.0093
5	**0.9370 ± 0.0063**	**0.9685 ± 0.0032**	0.6880 ± 0.0273	0.8411 ± 0.0136	0.8163 ± 0.0143	0.9082 ± 0.0071
10	**0.9359 ± 0.0085**	**0.9679 ± 0.0043**	0.6540 ± 0.0200	0.8270 ± 0.0100	0.8237 ± 0.0157	0.9118 ± 0.0082
15	**0.9358 ± 0.0089**	**0.9679 ± 0.0044**	0.6378 ± 0.0215	0.8189 ± 0.0108	0.8238 ± 0.0216	0.9119 ± 0.0108
20	**0.9363 ± 0.0067**	**0.9681 ± 0.0033**	0.6282 ± 0.0116	0.8141 ± 0.0058	0.8221 ± 0.0153	0.9111 ± 0.0077

Bold values correspond to best performances

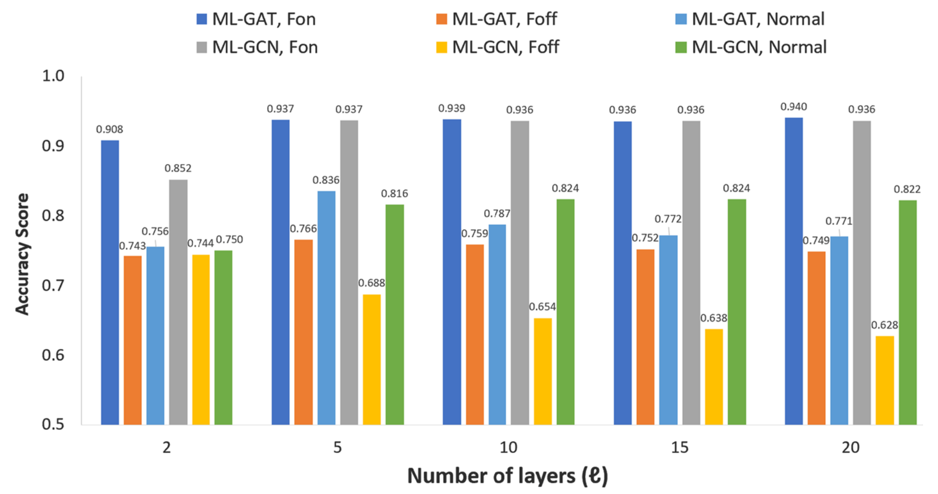


Fig. 7 Graphical representation of the performance of our framework on the Koumbia multilayer network

The time complexity of ML-GCN with K layers results from the addition of two terms, the one corresponding to the propagation steps, which is $\mathcal{O}(Knonzero(\mathbf{A}^{\text{sup}})f)$, where $nonzero(\mathbf{A}^{\text{sup}})$ is the number of non-zero entries in the $\mathbf{A}^{\text{sup}}$ matrix, and the other one corresponding to the feature transformation steps, which is $\mathcal{O}(KN\ell f^2)$. Therefore, the total cost of ML-GCN is $\mathcal{O}(Knonzero(\mathbf{A}^{\text{sup}})f + KN\ell f^2)$. The time complexity of ML-GAT also takes into account the computation of the attention coefficients. Given Q attention heads, the time complexity of ML-GAT with K layers is $\mathcal{O}(KQN\ell f^2 + KQ|E_{\mathcal{L}}|f)$, where the first term concerns the feature transformation steps, and the second term corresponds to the cost of a general attention mechanism. Note that the computation of the attention coefficients can be parallelized both for the intra-layer and inter-layer edges, as well as the computation of the Q attention heads.

Concerning the spatial complexity, modeling inter-layer dependencies in the propagation rule has the overhead of storing the whole supra-adjacency matrix. Moreover, we need to take into account the hidden states and the weight matrices. More precisely, the memory requirement during the training stage for ML-GCN is $\mathcal{O}(Kf^2 + KNf)$, whereas for the multi-head attention ML-GAT, this cost is multiplied by a factor Q. Furthermore, the attention functions value requires an overhead of $\mathcal{O}(Q|E_{\mathcal{L}}|)$. It is also worth noticing that, to improve scalability of our implementations, we could learn

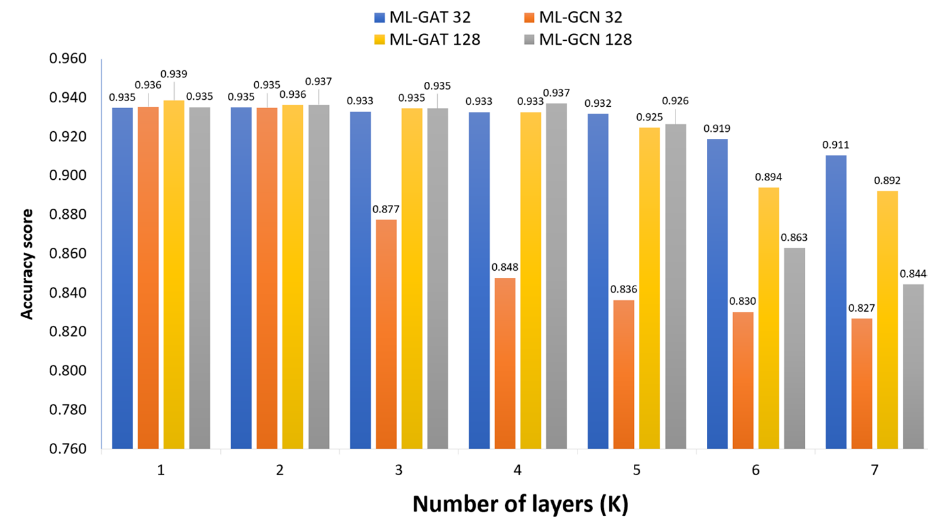


Fig. 8 Accuracy obtained by ML-GAT and ML-GCN, for increasing number of hidden layers *K*, and with $d = \{32, 128\}$

our models with mini-batch training in combination with neighborhood sampling approaches (e.g., Hamilton et al. 2018), which we leave it as for future work.

We now present the results reported in Table 10, which shows the training times obtained by the proposed ML-GAT and ML-GCN methods, compared to those by GrAMME-SG and GrAMME-Fusion. We observe that our methods are extremely efficient, especially ML-GCN, with training times under 0.3 minutes on all networks. Remarkably, similar training times on all networks are observed for ML-GAT and ML-GCN, respectively, thus hinting at their scalability. Both our methods significantly outperform GrAMME-SG and GrAMME-Fusion: indeed, except for *Vickers*, the training time of the GrAMME methods is always one or two orders of magnitude higher than that of our methods, also showing to have scalability issues (training times range from the 0.3 minutes of *Vickers* to 52 minutes on *Koumbia* for GrAMME-SG and GrAMME-Fusion, respectively).

The outperforming behavior of our methods against GrAMME ones is however quite surprising, since all the methods share a theoretical computational complexity that is linear in the number of nodes and in the number of edges of the multilayer network. In fact, as reported in Shanthamallu et al. (2020), while the cost of GrAMME-SG is actually more sensitive to the number of entities and layers in the network (i.e., linear in the number of entities and edges, but quadratic in the number of layers), the cost of GrAMME-Fusion, thanks to a simplified attention mechanism, is declared as linear in the number of entities and edges of the multilayer network, which is analogous for our methods. Therefore, we tend to ascribe such a performance gap of the competitors to a less efficient implementation w.r.t. our methods, which were developed under the DGL framework[6] that has become a widely recognized software tool for deep learning on graph data (Wang et al. 2020).

[6] https://www.dgl.ai.

Evaluation on real-world node-attributes and arbitrary inter-layer edges: the Koumbia multilayer network testbed

A major strong point of the proposed ML-GAT and ML-GCN approaches is that they are designed to deal with general multilayer networks, i.e., with arbitrary inter-layer edges, and to exploit external information in the form of attributes associated to the entities. In this section, we present a further evaluation stage that aims to stress our methods by evaluating them on a real-world attributed multilayer network, i.e., the *Koumbia* multilayer network (Interdonato et al. 2020). By focusing on this network, we delve into the understanding of the impact of using real-world attributes about the entities (cf. "Data" section and Fig. 5a) on the entity classification task in a practical application contexts. Moreover, based on the technique described in Interdonato et al. (2020), we take into account different versions of the network with varying number of layers (i.e., 2, 5, 10, 15, 20) and, since the networks include inter-layer edges between each couple of layers, we will also evaluate how the proposed approach is able to manage an increasing number of inter-layer edges.

To this purpose, we compare the performance of our methods on three different scenarios relating the input features associated with various *Koumbia* with different number of layers: the real attributes originally associated with *Koumbia* entities, attributes in the form of identity matrix, and attributes modeled as normal distributions. The three modalities will be denoted with suffix *Fon*, *Foff*, and *normal*, respectively. Note that the experiments with the normal distributions have different results with respect to the ones reported in Table 5, since in that case the multiplex version of the network was taken into account (i.e., without considering inter-layer edges).

Tables 11 and 12 show the average accuracy and mean reciprocal rank (MRR), averaged over 20 runs, obtained on the *Koumbia* networks with *Fon*, *Foff* and *normal* types of attributes, for ML-GAT and ML-GCN, respectively. A detailed plot on the variations of accuracy with respect to the number of layers is also shown in Fig. 7. It can be noted that the *Fon* versions always obtain significantly better result than the *Foff* and *normal* ones for both methods, thus confirming that exploiting real-world node-attributes is indeed beneficial for the entity classification task and, more importantly, that the proposed framework is able to correctly exploit such external information in the form of node attributes.

ML-GAT and ML-GCN obtain similar performances on the *Fon* networks (accuracy around 0.94 and MRR around 0.97). The performance scores also show robustness with respect to the number of layers, and hence of inter-layer edges, in the network. Note that the slightly lower performance obtained for *Koumbia*-2 is actually not surprising, as in that case the two layers have disjoint node-sets, which negatively affects the performance of the multilayer approaches.

It is also interesting to notice that, while ML-GAT performs similarly on the *Foff* and *normal* versions (thanks to the attention mechanism), ML-GCN shows significantly better performance on the *normal* version than on the *Foff* one. This result is in line with previous studies (Kipf and Welling 2017; Velickovic et al. 2018), and, in this specific case, it can also be explained by the fact that the normal distribution can be a relatively good approximation of the real one (cf. Fig. 5).

In order to further analyze the benefit of the attention mechanism exploited by ML-GAT, against the convolutional approach used in ML-GCN, we perform a further analysis stage, where we evaluate the performance of the two approaches w.r.t. an increasing number of hidden layers K in the neural network (i.e., not to be confused with the layers of the multilayer network). Figure 8 shows the accuracy achieved by ML-GAT and ML-GCN by increasing K, and with different dimensions of the embedding space, i.e., $d = \{32, 128\}$. It can be noted that, while for $K = \{1, 2\}$ all methods obtain similar performance, ML-GCN tends to decrease in accuracy for higher K values; particularly, ML-GCN accuracy decreases of about 7% when increasing K from 2 to 3, with $d = 32$, and from 5 to 6 with $d = 128$. We tend to explain this behavior since a higher number of convolutional layers would smooth the difference between intra-layer and inter-layer neighborhoods, which hence might be treated equally in this process. Conversely, the attention mechanism in ML-GAT is way more robust to this phenomenon, as revealed by the nearly constant performance by ML-GAT even with high values of K.

Conclusions: discussion and future work

We proposed a GNN framework for representation learning and semi-supervised classification in multilayer networks with attributed entities, and with arbitrary number of layers and intra-layer and inter-layer connections between nodes. We instantiated our framework through two new formulations of GAT and GCN models, specifically designed for the above general, attributed multilayer networks. We evaluated our ML-GAT and ML-GCN methods on real-world network datasets coming from different domains and with different structural characteristics. Our results showed that ML-GAT and ML-GCN models are significantly faster learners than the competitors, and they outperform in accuracy both the competitors and baseline methods especially on arbitrary multilayer networks, with large number of entities and layers. Furthermore, as demonstrated by the evaluation on *Koumbia* multilayer networks, derived from satellite images, our methods are able to take advantage of the presence of real attributes for the entities, in addition to arbitrary inter-layer connections between the nodes in the various layers.

Comparing the GAT and GCN approaches, we observed that, unlike ML-GCN, ML-GAT performance is not affected when networks are structured as disconnected layers or when most layers tend to contain nodes of the same label. By contrast, ML-GCN tends to be more robust than ML-GAT when the network shows relatively high density and average degree.

Nevertheless, the approach of integrating within-layer and outside-layer neighborhood shared by both ML-GCN and ML-GAT might not be well-suited to effectively learn from multilayer networks where the various layers would show assortativity different to each other according to the entity class labels; e.g., a 2-layer network with gender as entity class, such that the first layer is assortative by gender and the second layer shows reverse assortativity by gender. To overcome this limitation, it would be interesting to revise the cross-layer aggregation component in terms of a GNN model as well, and investigate how this approach would be more effective than simply weighing the embeddings from each particular layer of the network. As a related aspect, the above would also raise the opportunity of evaluating a transfer learning task across the layers of a network: for instance, given one or more selected layers, our methods would train a model on those layers which would then be fine-tuned on other layers, e.g., for a task of node classification. Moreover, measuring the similarity between the different layers of a multilayer network (e.g., via the subspace alignment measure proposed in Qian et al. 2021), and more in general, multilayer network simplification approaches (Interdonato et al. 2020) could be consider in order to improve the quality of the final embeddings by reducing the quantity of redundant or noisy content in each layer.

Further developments of our framework might concern two aspects. The first aspect refers to an extension to *heterogeneous*, attributed multilayer networks, which are rapidly growing attention also thanks to a renewed interest to knowledge graphs in many application domains. The second aspect instead refers to the adaptation of our framework to *inductive* learning tasks, in order to generalize to unseen (portions of) graphs. This would also enable it to deal with *dynamic networks*, particularly for understanding the growth of a multilayer network in terms of changes on the status and properties of its entities and their connections, or for updating a GNN model on a time-evolving multilayer network without having to learn it from scratch at each new timestamp. In this regard, while the adaptation of our defined methods' propagation rules is relatively easy to achieve for an inductive learning task, it would also be meaningful to introduce an unsupervised term in the loss function as a form of regularization to account for the structural information of the unseen portions of a network.

Abbreviations
GNN: Graph Neural Network; GCN: Graph Convolutional Network; GAT: Graph Attention Network; ML-GCN: Multilayer Graph Convolutional Network; ML-GAT: Multilayer Graph Attention Network; CNN: Convolutional Neural Network; RNN: Recurrent Neural Network.

Acknowledgements
Not applicable.

Authors' contributions
AT conceived the idea presented in this work. AT and LZ developed the theoretical definition of the methods. AT and RI defined the set of experiments to perform. LZ took care of running the experiments. RI, LZ, and AC performed evaluation of the results and related discussion. All authors participated in the writing process. All authors read and approved the final manuscript.

Funding
This work was supported by the French National Centre for Space Studies (CNES), as part of the AMORIS (Analyse et MOdélisation des Réseaux complexes issus de l'Imagerie Satellitaire) project, APR TOSCA 2020.

Availability of data and materials
Python code for the proposed methods, as well as the network datasets, will be made publicly available upon publication of the manuscript.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Consent for publication
All authors read and approved the final manuscript.

Author details
[1]Department of Computer Engineering, Modeling, Electronics, and Systems Engineering (DIMES), University of Calabria, Rende, Italy. [2]UMR TETIS, Cirad, Montpellier, France. [3]TETIS, AgroParisTech, CIRAD, CNRS, INRAE, Univ Montpellier, Montpellier, France.

Received: 30 April 2021 Accepted: 30 August 2021
Published online: 08 November 2021

References

Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P (2017) Geometric deep learning: going beyond Euclidean data. IEEE Signal Process Mag 34(4):18–42
Chen P, III AOH (2016) Multilayer spectral graph clustering via convex layer aggregation. In: Proceedings of IEEE global conference on signal and information processing, pp 317–321
Coleman J, Katz E, Menzel H (1957) The diffusion of an innovation among physicians. Sociometry 20(4):253–270
Everton SF (2012) The Noordin top terrorist network. In: Disrupting dark networks. Structural analysis in the social sciences. Cambridge University Press, Cambridge, pp 385–396. https://doi.org/10.1017/CBO9781139136877.019
Gaito S, Interdonato R, Murata T, Sala A, Tagarelli A, Thai MT (2021) Introduction to the special section on reloading feature-rich information networks. IEEE Trans Netw Sci Eng. https://doi.org/10.1109/TNSE.2021.3073824
Ghorbani M, Baghshah MS, Rabiee HR (2019) MGCN: semi-supervised classification in multi-layer graphs with graph convolutional networks. In: Proceedings of IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM), pp 208–211. https://doi.org/10.1145/3341161.3342942
Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE (2017) Neural message passing for quantum chemistry. In: Proceedings of 34th international conference on machine learning, pp 1263–1272
Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of thirteenth international conference on artificial intelligence and statistics, pp 249–256
Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 855–864
Hamilton WL, Ying R, Leskovec J (2018) Inductive representation learning on large graphs. CoRR arXiv:abs/1706.02216arXiv:1706.02216
Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
Interdonato R, Atzmueller M, Gaito S, Kanawati R, Largeron C, Sala A (2019) Feature-rich networks: going beyond complex network topologies. Appl Netw Sci 4(1):4–1413
Interdonato R, Gaetano R, Lo Seen D, Roche M, Scarpa G (2020) Extracting multilayer networks from sentinel-2 satellite image time series. Netw Sci 8(S1):26–42. https://doi.org/10.1017/nws.2019.58
Interdonato R, Magnani M, Perna D, Tagarelli A, Vega D (2020) Multilayer network simplification: approaches, models and methods. Comput Sci Rev 36:100246
Kingma DP, Ba J (2017) Adam: a method for stochastic optimization. CoRR arXiv:abs/1412.6980
Kipf TN, Welling M (2016) Variational graph auto-encoders. CoRR arXiv:abs/1611.07308
Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of 5th international conference on learning representations (ICLR)
Kivelä M, Arenas A, Barthelemy M, Gleeson JP, Moreno Y, Porter MA (2014) Multilayer networks. J Complex Netw 2(3):203–271
LeCun Y, Bengio Y (1995) Convolutional networks for images, speech, and time-series. In: Arbib MA (ed) The handbook of brain theory and neural networks. MIT Press, Cambridge
LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444. https://doi.org/10.1038/nature14539
Li J, Chen C, Tong H, Liu H (2018) Multi-layered network embedding. In: Proceedings of SIAM international conference on data mining (SDM), pp 684–692. https://doi.org/10.1137/1.9781611975321.77
Liu W, Chen P-Y, Yeung S, Suzumura T, Chen L (2017) Principled multilayer network embedding. CoRR arXiv:abs/1709.03551
Magnani M, Hanteer O, Interdonato R, Rossi L, Tagarelli A (2021) Community detection in multiplex networks. ACM Comput Surv 54(3):48–14835. https://doi.org/10.1145/3444688
Ma Y, Liu X, Shah N, Tang J (2021) Is homophily a necessity for graph neural networks? arXiv:2106.06134
Mishkin D, Matas J (2016) All you need is a good init. In: Proceedings of international conference on learning representations (ICLR). arXiv:1511.06422
Mohan APKV (2021) Temporal network embedding using graph attention network. Complex Intell Syst. https://doi.org/10.1007/s40747-021-00332-x
Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Macskassy SA, Perlich C, Leskovec J, Wang W, Ghani R (eds) Proceedings of 20th ACM SIGKDD international conference on knowledge discovery and data mining, pp 701–710
Qian Y, Expert P, Rieu T, Panzarasa P, Barahona M (2021) Quantifying the alignment of graph and features in deep learning. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/tnnls.2020.3043196
Ross T (2009) Fuzzy logic with engineering applications, 3rd edn. Wiley, Hoboken. https://doi.org/10.1002/9781119994374
Schlimmer JC (1987) Concept acquisition through representational adjustment
Shanthamallu US, Thiagarajan JJ, Song H, Spanias A (2020) GrAMME: semisupervised learning using multilayered graph attention models. IEEE Trans Neural Netw Learn Syst 31(10):3977–3988. https://doi.org/10.1109/TNNLS.2019.2948797
Siegler RS (1976) Three aspects of cognitive development. Cogn Psychol 8(4):481–520
van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9:2579–2605
Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Lu, Polosukhin I (2017) Attention is all you need. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in neural information processing systems, vol 30
Velickovic P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks. In: Proceedings of 6th international conference on learning representations (ICLR)
Vickers M, Chan S (1981) Representing classroom social structure. Victoria Institute of Secondary Education, Melbourne

Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol P-A (2010) Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J Mach Learn Res 11:3371–3408
Wang M, Zheng D, Ye Z, Gan Q, Li M, Song X, Zhou J, Ma C, Yu L, Gai Y, Xiao T, He T, Karypis G, Li J, Zhang Z (2020) Deep graph library: a graph-centric, highly-performant package for graph neural networks. CoRR abs/1909.01315
Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS (2021) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst 32(1):4–24. https://doi.org/10.1109/tnnls.2020.2978386
Xu K, Hu W, Leskovec J, Jegelka S (2019) How powerful are graph neural networks? In: Proceedings of 7th international conference on learning representations (ICLR)
Xu K, Li C, Tian Y, Sonobe T, Kawarabayashi K-I, Jegelka S (2018) Representation learning on graphs with jumping knowledge networks. arXiv:1806.03536
Zhou J, Cui G, Zhang Z, Yang C, Liu Z, Wang L, Li C, Sun M (2019) Graph neural networks: a review of methods and applications. CoRR arXiv:abs/1812.08434
Zhu J, Yan Y, Zhao L, Heimann M, Akoglu L, Koutra D (2020) Beyond homophily in graph neural networks: current limitations and effective designs. In: Proceedings of the annual conference on neural information processing systems (NeurIPS)

Publisher's Note