



# A network analysis-based framework to understand the representation dynamics of graph neural networks

Gianluca Bonifazi<sup>1</sup> · Francesco Cauteruccio<sup>1</sup> · Enrico Corradini<sup>1</sup> · Michele Marchetti<sup>1</sup> · Domenico Ursino<sup>1</sup>  · Luca Virgili<sup>1</sup>

Received: 17 May 2023 / Accepted: 20 October 2023 / Published online: 17 November 2023  
© The Author(s) 2023

## Abstract

In this paper, we propose a framework that uses the theory and techniques of (Social) Network Analysis to investigate the learned representations of a Graph Neural Network (GNN, for short). Our framework receives a graph as input and passes it to the GNN to be investigated, which returns suitable node embeddings. These are used to derive insights on the behavior of the GNN through the application of (Social) Network Analysis theory and techniques. The insights thus obtained are employed to define a new training loss function, which takes into account the differences between the graph received as input by the GNN and the one reconstructed from the node embeddings returned by it. This measure is finally used to improve the performance of the GNN. In addition to describe the framework in detail and compare it with related literature, we present an extensive experimental campaign that we conducted to validate the quality of the results obtained.

**Keywords** Graph neural networks · Social network analysis · Learned representations · Node embeddings · Loss function

## 1 Introduction

Graph Neural Networks (hereafter, GNNs) have received a significant attention in recent years for their ability to work with graph-structured data. They are designed to learn node and arc representations in a graph, capable of capturing important features of the underlying data [1–3]. In

particular, GNNs are very successful in tasks where graph structure plays a key role in determining the output [4, 5], for example in identifying communities within a social network, predicting links between nodes, classifying nodes or graphs [6–9]. These tasks have important applications in several domains, such as Social Network Analysis, bioinformatics and recommender systems [6, 10, 11], materials science and chemistry [12]. Studies on challenging tasks, such as protein-protein interaction [13] and causal inference in brain networks [14], have demonstrated the expressive power and potential of GNNs and stimulated further investigation by the academic community.

The growing popularity of GNNs has led to the development of various architectures and algorithms related to them [15–18], as well as of approaches to improve their discriminative power [19, 20]. However, despite their success, there are still many open questions about how GNNs learn and represent information. One of the challenging issues is to understand the dynamics underlying learned representations and how these representations relate to the underlying graph structure. For example, it is unclear how GNNs process different types of information (e.g., node attributes and arc features) in their learned

---

✉ Domenico Ursino  
d.ursino@univpm.it

Gianluca Bonifazi  
g.bonifazi@univpm.it

Francesco Cauteruccio  
f.cauteruccio@univpm.it

Enrico Corradini  
e.corradini@univpm.it

Michele Marchetti  
m.marchetti@pm.univpm.it

Luca Virgili  
luca.virgili@univpm.it

<sup>1</sup> Department of Information Engineering (DII), Polytechnic University of Marche, 60131 Via Breccie Bianche, 12, Ancona, Italy

representations, and how they capture different types of structural patterns in graph data. Another challenge concerns the development of methods to evaluate the quality and generalization of learned representations in GNNs. This is particularly important in real-world applications, where GNNs must be robust and generalize accurately to unknown data. Finally, understanding the limitations and biases of GNNs can help identify potential improvement areas and provide insights into the design of better models capable of solving complex graph-based problems.

In this scenario, a compelling and interesting aspect regards the study of learned representations, especially node embeddings, as it provides insights into how GNNs capture and process information in a graph. Node embeddings model each node in a graph as a vector representing the corresponding structural and semantic information. By examining how GNNs handle the information present in input graphs, we can obtain valuable insights into their encoding and processing mechanisms. This analysis can also help us understand how GNNs generalize this information to new data. Moreover, node embeddings can be used in a wide range of specific tasks (think, for instance, of node classification, link prediction and graph clustering), which represent fundamental problems in areas like Network Analysis (NA, for short) and Machine Learning.

This paper falls right in this context and proposes a comprehensive framework that uses (Social) Network Analysis theory and techniques to investigate learned representations of GNNs. NA theory provides a powerful and intuitive means of analyzing the structure and properties of graph-structured data. When applied to learned representations of GNNs, this theory and the corresponding techniques can support the extraction of important insights enabling a deeper understanding of learned information. Furthermore, NA can support us in assessing the quality of learned representations and identifying improvement areas.

Our framework operates as follows: it receives a graph and passes it to the underlying GNN to be investigated. The GNN returns a set of node embeddings corresponding to the graph received as input. At this point, our framework creates a new network-based representation by combining the node embeddings returned by the GNN and the graph received as input. After that, it uses NA theory and techniques to obtain insights regarding the GNN performance. Then, it uses these insights to define a new loss function in order to enhance the training of the GNN. For this purpose, it considers the differences between the graph received as input and the one reconstructed from the node embeddings returned by the underlying GNN.

To the best of our knowledge, this paper represents one of the first attempts to study the learned representations of a GNN and to improve its learning process through a comprehensive framework based on NA. Specifically, the main

contributions of the framework presented in this paper are as follows:

- It proposes a method for mapping into a network the learned representations returned by a GNN after processing a graph received in input;
- It employs NA theory and techniques to extract insights about the structure and behavior of the underlying GNN;
- It defines a new training loss function to both assess and enhance the quality of the GNN's learning process, and thus the likelihood that it will next provide more accurate results.

The outline of this paper is as follows: Sect. 2 describes related literature. Section 3 presents the proposed framework. Section 4 illustrates the experimental campaign we conducted to test it. Finally, Sect. 5 reports our conclusion and outlines some possible future developments of our research efforts.

## 2 Related literature

GNNs have emerged as a powerful framework for learning representations of graph-structured data [1, 6]. Indeed, they are capable of encoding both node attributes and input network topology into node embeddings, which are vector representations of nodes capable of capturing their structural and/or semantic properties. A very active research area concerns the analysis of such embeddings to answer questions such as: “Do they capture meaningful relationships?”, “Can they be measured quantitatively?”. Questions like these can be addressed by studying the representational power of GNNs [21–23]. A large number of papers in the literature have shown that GNNs are an effective tool for learning graph representations in many applications. However, there is still little understanding about their properties, limitations and learned representations [21, 24]. This paper aims to make a contribution to filling this gap by investigating and providing insights into the dynamics underlying the learned representations returned by GNNs. To this end, it proposes a conceptual framework that employs the theory and techniques of Network Analysis to evaluate these learned representations.

As far as the scientific literature is concerned, the study proposed in this paper belongs to the broader field of representation learning through GNNs. In the following we will focus on two aspects of our paper, namely: (i) the study and analysis of learned representations in GNNs, and (ii) the use of (Social) Network Analysis models and techniques to investigate the dynamics of GNNs and assess the quality of their learned representations. With regard to

these aspects, we highlight what is already in the literature and how our framework relates to existing ones. In addition, in order to highlight the features of our framework, at the end of our overview we briefly discuss some application contexts that can benefit from the exploitation of GNNs and learned representations. We believe it is worth pointing out that our framework represents one of the first attempts to develop an approach that uses NA not only to evaluate the quality of learned representations in GNNs but also to enhance their training.

As for the study and analysis of GNNs learned representations, several surveys have recently been proposed [1, 6, 25]. The study in [21] is widely recognized as one of the most comprehensive studies on the expressive power of GNNs. In it, the authors present a theoretical framework for analyzing the expressive power of GNNs and apply it to characterize popular variants of GNNs, such as Graph Convolutional Networks. They also develop a simple neural architecture, called Graph Isomorphism Network, which has been shown to be as powerful as other GNNs. The work in [21] and ours are related in terms of motivations. Indeed, the work in [21] suggests that a maximally powerful GNN should be able to distinguish different graph structures by mapping them to different representations in the embedding space, but this is a hard problem. Our approach addresses this issue by analyzing the learned representations of a GNN, using the theory and techniques of NA, which intrinsically encompasses properties of graph structures.

In [26], the authors use spectral analysis to investigate the expressive power of GNNs. They argue that this perspective provides a complementary viewpoint for understanding GNNs and demonstrate the equivalence of convolution processes in spatial and spectral GNNs. Moreover, through several experiments, they show that graph convolutions in GNNs are problem-specific rather than problem-agnostic. The approach in [26] is orthogonal to ours since it focuses on characterizing the expressive power of GNNs from a spectral perspective while our approach focuses on the analysis of learned representations. In [27], the authors propose an interpretable embedding procedure based on a knowledge distillation method that leverages the learned representation of a GNN. The authors show that the graph structure caught in the learned representation captures relational information better than classical representations, such as those obtained from attention networks. The approach in [27] and ours share the use of GNNs learned representation. However, they have different goals. In fact, the approach in [27] focuses on knowledge distillation and wants to generate an interpretable embedding procedure. By contrast, our approach focuses on the investigation of embeddings through the theory and techniques of NA.

In [28], the authors focus on the graph comparison problem and propose an approach to produce a fast-to-compute feature map that represents a graph through the distribution of its node embeddings. The authors employ the proposed approach in a graph classification task and compare it with other supervised and unsupervised approaches. This paper is interesting in that the authors explore embedding techniques and construct particular feature maps on top of the results, thus effectively exploiting learned representations. However, unlike our approach, the one of [28] does not consider learned representations obtained by GNNs. In [29], the authors investigate the empirical robustness of embeddings, produced by different models, to random and adversarial poisoning attacks. In their evaluation, they include matrix factorization-based models, skip-gram-based models, and deep neural network-based models. Although both our approach and that of [29] analyze learned representations, the approach in [29] is devoted to a different task, namely assessing a specific property of node embeddings.

In [30], the authors present ROLAND, a graph representation learning framework for real-world dynamic graphs. ROLAND allows researchers to easily repurpose any static GNN to dynamic graphs. The idea is to view node embeddings at different GNN layers as hierarchical node states and then recurrently update them over time. While both the approach in [30] and ours leverage learned representations, they have very different goals. Nevertheless, in future, it would be interesting to conduct our study on dynamic graphs also exploiting models built by ROLAND.

As for the analysis of GNN dynamics and the evaluation of the quality of their learned representations, in [31] the authors discuss the representation power of Graph Convolutional Networks (GCNs) in learning graph moments, which encode paths of various lengths in graph topology. The approach in [31] and ours share the idea of analyzing learned representations through graph properties and structures. However, the approach in [31] aims to prove the limitations of a GCN in learning graph topology, particularly graph moments, while our approach studies learned representations directly through the proposed framework. Moreover, the approach in [31] targets generation models while ours focuses on learned representations of any GNN. In [32], the authors discuss the limitations in using random walk-based sampling strategies for network embeddings. They also present a new method that combines neighbor information and local-subgraph similarity to learn node embeddings. This method also uses structural information (graphlets) to enhance the quality of embeddings. It aims to enhance learned representations by using topological information for network data. However, it does not perform

a subsequent analysis of the enhanced learned representation.

In [33], the authors propose NCA-GE (Network Centrality Approximation using Graph Embeddings), a fast and efficient approach to approximate node centralities in large networks using neural networks and graph embedding techniques. NCA-GE represents a direct application of learned representations for addressing a task that cannot be performed easily on very large graphs. NCA-GE and our framework can be considered complementary. In fact, our framework could exploit the results of NCA-GE to further measure the quality of learned representations. In [34], the authors propose a new approach called DEMO-Net. It performs multi-task graph convolution, where each task carries out node representation learning for nodes with a specific degree value. The authors also introduce a new graph-level pooling/readout scheme for learning graph representations, and show that this scheme is efficient and effective in many cases. DEMO-Net also focuses on modifying the learning model to explicitly capture the graph topology through a variant of GNN.

In [35], the authors investigate structural node embeddings. These embeddings are based on the principle that nodes having similar functions, ties or interactions should be close in the embedding space, regardless of their distance in the network. In the paper, the authors want to understand what types of equivalence are captured by structural embeddings and provide an in-depth empirical analysis of them using a variety of datasets and tasks. The approach in [35] and ours share some similarities. In fact, both of them propose an intrinsic evaluation of node embeddings and in both cases this evaluation is based on some network parameters. However, the approach in [35] focuses on structural node embeddings while our framework can be applied to any type of node embeddings. In addition, our framework also wants to improve the learning process of GNNs. In [36], the authors aim to interpret vector embeddings of social network data and propose concrete interpretations in terms of preserved network properties. Their approach relates embeddings with network centralities. As a result, they obtain that different embedding methods learn different network properties. Both the approach in [36] and our framework use (Social) Network Analysis to provide an explanation of embedding relatedness. However, their goals are different. In fact, the approach in [36] wants to predict the centrality values of a particular node based on its embeddings. In contrast, our framework focuses on analyzing the structures within the embeddings and improving the learning process. Finally, the approach in [36] does not consider the learned representations resulting from GNNs.

In [37], the authors analyze the effectiveness of vector embeddings of nodes in encoding the elementary properties

of the nodes themselves. They also evaluate three state-of-the-art node representation models (i.e., DeepWalk, node2vec and LINE) on different tasks and different graphs, and show that node2vec and LINE best encode network properties for sparse and dense graphs, respectively. The approach in [37] and our framework share the use of NA to evaluate node embeddings. However, the goals of the two approaches are different. In fact, the approach in [37] focuses on the effectiveness of various node representations in predicting the properties of graphs and builds a model to do this. In contrast, our approach wants to evaluate node representations and use that evaluation to improve GNN learning. Furthermore, the approach in [37] does not consider learned representations resulting from GNNs, but uses classical models, such as DeepWalk. In [38], the authors propose a framework for unsupervised graph embedding comparison. Although the goals of this approach differ from those of our framework, some concepts and insights present in [38] (e.g., the concept of divergence score) may be useful in our context.

To conclude, we point out that the expressive power of GNNs enabled a series of contexts and tasks to be successfully managed by them. In particular, researchers have investigated the usage of GNNs to solve problems in complex and social networks [39–42]. The ability of extracting node-level structural features from graphs is one of the key aspects of GNNs. For instance, in [39], the authors propose a GCN-based framework for the estimation of communication network reliability. This framework employs several graph convolution layers to extract node-level structural features from input information. The crucial task of identifying critical nodes and links in graphs is the focus of the approach described in [40]. Here, the authors propose a scalable and generic GNN for identifying critical nodes and links in large complex networks. The idea is to learn the node and link criticality score on a small representative subset of nodes and links, and then predict the scores of nodes and links on a larger scale network. An approach tackling general time-evolving social network problems is presented in [41]. Here, the authors propose a GNN-based framework, called Spatial-Temporal Graph Social Network (STGSN), whose aim is to model a social network taking both spatial and temporal perspectives into account. STGSN belongs to the general field of GNNs supporting Social Network Analysis; therefore, it can be considered orthogonal to our approach. In fact, while STGSN employs GNNs for modeling the dynamics of a time-evolving Social Network, our framework employs (Social) Network Analysis models and techniques to investigate the dynamics of GNNs. To address the social recommendation problem, the authors of [42] propose a GNN-based framework able to coherently model graph data with the aim to learn better user and item

representations. The framework takes advantage of an attention mechanism to discern the heterogeneous strengths of social relationships between users. Also the study of functional brain networks has been addressed through GNNs. For instance, in [43] the authors propose BrainTGL, a temporal graph representation learning framework for brain networks. The objective is to capture the potentially complex spatial and temporal correlations in human brain through a combination of approaches, such as temporal graph pooling and dual temporal graph learning.

### 3 Description of our framework

In this section, we present our NA-based framework conceived to investigate the node embeddings returned by a GNN and to improve its learning process. Specifically, in Sect. 3.1 we define the network model adopted by our framework. In Sect. 3.2, we describe a set of analysis measures used by it to achieve its goals. Finally, in Sect. 3.3 we describe the GNN evaluation and enhancement process performed by it.

#### 3.1 Description of the model underlying our framework

Let  $G$  be a graph to be processed and let  $\mathcal{GNN}$  be a Graph Neural Network that performs a machine learning task (e.g., node classification, graph classification, edge prediction, etc.) on  $G$ . The ultimate goal of our framework is to evaluate the application of  $\mathcal{GNN}$  to  $G$  and improve its performance.

$G$  can be modeled as  $G = \langle V, E, W \rangle$ . Here,  $V$  is the set of nodes; each node  $v_i \in V$  has associated a set  $X_i \in \mathbb{R}^h$ ,  $h \geq 1$ , of features.  $E \in V \times V$  is the set of edges. An edge  $e_{ij} \in E$  exists between the nodes  $v_i$  and  $v_j$  if there exists a relationship<sup>1</sup> between them. There exists a weight  $w_{ij} \in W$  for each edge  $e_{ij} \in E$ ; it indicates the strength of the relationship between  $v_i$  and  $v_j$ .

This modeling is as general as possible. Therefore, depending on the specific application,  $v_i$ ,  $X_i$  and  $e_{ij}$  have different meanings. For example, in the well-known Cora dataset,<sup>2</sup> each node  $v_i$  represents a paper.  $X_i$  is a one-hot encoded vector such that each of its elements is associated with a word of interest and is set to 1 if the associated word is present in the paper corresponding to  $v_i$ , otherwise it is set to 0. The arc  $e_{ij}$  indicates a relationship between  $v_i$  and  $v_j$  such that the paper corresponding to  $v_i$  cites the one associated with  $v_j$ . The corresponding weight  $w_{ij}$  is always

equal to 1. In this example,  $G$  is a directed graph, but the model we are defining is generic and admits both directed and undirected graphs.

Suppose we train  $\mathcal{GNN}$  for  $\eta$  epochs. During its training,  $\mathcal{GNN}$  learns the node representations in a latent space, i.e., node embeddings. We can define a function  $f: \mathbb{R}^h \rightarrow \mathbb{R}^{emb}$  representing this learning task. Here,  $h$  is the number of features of a node,  $emb$  is the dimension of the embedding; generally,  $emb \ll h$ . In particular, given a node  $v_i$  of  $G$  and the feature vector  $X_i$  of  $v_i$ , the embedding  $\mathcal{E}_i$  of  $v_i$  is a vector in  $\mathbb{R}^{emb}$ . It captures both structural and feature-based properties of  $v_i$  because it derives directly from the learning activity of  $\mathcal{GNN}$  performed on  $G$  and the vector  $X_i$  of the features of  $v_i$ .

At the end of each training epoch  $t$ , we can use  $f$  to obtain the corresponding node embeddings at that time in order to analyze the evolution of learned representations and to enhance the training of  $\mathcal{GNN}$ . To address this issue we need a way to map the embeddings back to the original graph. For this purpose, we introduce a new graph  $G^e(t) = \langle V, E, W^e(t) \rangle$ , which represents a variant of  $G$ . In fact,  $G^e(t)$  has the same set  $V$  of nodes and the same set  $E$  of edges of  $G$ , while the vector  $W^e(t)$  of the edge weights is different. In fact, since  $G^e(t)$  contains feature vectors different from those of  $G$ , it is necessary to take this additional information into account, and this is done precisely by redefining  $W^e(t)$ . One way to address this issue is to consider the distance and/or similarity between the embeddings of the two nodes associated with the edges whose weight is being computed. Several measures capable of doing this have been proposed in the literature. One of the most widely used is cosine similarity, which computes the cosine of the angle between the two vectors. It ranges between -1 and 1; in particular, it is set to 1 if the two vectors are exactly the same, to 0 if they are orthogonal and to -1 if they are diametrically opposed. Taking this similarity into account, the weight  $w_{ij}^e(t)$  associated with the edge  $e_{ij}$  in  $G^e(t)$  can be computed as follows:

$$w_{ij}^e(t) = w_{ij} \frac{\mathcal{E}_i \cdot \mathcal{E}_j}{\|\mathcal{E}_i\| \|\mathcal{E}_j\|} = w_{ij} \frac{\sum_{k=1}^{emb} \mathcal{E}_{i_k} \mathcal{E}_{j_k}}{\sqrt{\sum_{k=1}^{emb} \mathcal{E}_{i_k}^2} \sqrt{\sum_{k=1}^{emb} \mathcal{E}_{j_k}^2}} \quad (3.1)$$

Here,  $\mathcal{E}_i$  and  $\mathcal{E}_j$  are the embedding vectors of  $v_i$  and  $v_j$  in  $G^e(t)$ , while  $w_{ij}$  is the weight of the edge  $e_{ij}$  in the original graph  $G$ . The reasoning behind this formula is as follows: the weight  $w_{ij}^e(t)$  is obtained by multiplying the weight of the original edge  $w_{ij}$  by the cosine similarity of the embedding vectors associated with  $v_i$  and  $v_j$  in  $G^e(t)$ . In this way, the initial weight  $w_{ij}$  varies by a factor defined by

<sup>1</sup> At this moment, it is not important to indicate the specific relationship between the two nodes of an edge.

<sup>2</sup> <https://relational.fit.cvut.cz/dataset/CORA>.

the embeddings extracted by  $\mathcal{GNN}$ . Thus, the strength of the connections between the nodes in  $G_{ij}^e(t)$  changes according to the training of  $\mathcal{GNN}$ . We point out that this formula also allows the edges of  $G_{ij}^e(t)$  to maintain the same semantics as the edges of  $G$  since the weights are only scaled by a pure factor.

### 3.2 Analysis measures used by our framework

In this section, we present the analysis measures employed by our framework to reach its goals. Specifically, in Sect. 3.2.1 we describe two measures to compare communities in  $G$  and in  $G^e(t)$ . In Sect. 3.2.2, we present a measure to analyze the clustering coefficient of  $G^e(t)$  during the training of  $\mathcal{GNN}$ . Finally, in Sect. 3.2.3, we illustrate a measure to evaluate the differences between  $G$  and  $G^e(t)$  in terms of centrality measures.

#### 3.2.1 Two measures for comparing communities in $G$ and $G^e(t)$

In this section, we want to study the structure of the communities that can be derived from  $G$  and  $G^e(t)$ . In particular, we propose two ways to perform such a task. Preliminarily, we need an algorithm  $\mathcal{A}_{CD}$  for community detection on graphs with weighted edges, such as Louvain [44], FastGreedy [45], Label Propagation Algorithm [46], and/or another among those proposed in the past literature [47]. The application of  $\mathcal{A}_{CD}$  to  $G$  and  $G^e(t)$  returns two sets of communities  $\mathcal{CS}$  and  $\mathcal{CS}^e(t)$ . Once we have these two sets, we need a way to identify which among them best splits the corresponding graph in communities characterized by many strong intra-community edges and few weak inter-community edges. A popular metric addressing this issue is modularity [48], defined as:

$$Q = \frac{1}{2w_{tot}} \sum_{i,j=1}^n \left[ A[i,j] - \frac{d_i d_j}{2w_{tot}} \right] \delta(\mathcal{C}_i, \mathcal{C}_j) \quad (3.2)$$

Here: (i)  $w_{tot}$  is the sum of the weights of the edges in the network into consideration; (ii)  $A[i, j]$  is the element at position  $(i, j)$  of the adjacency matrix corresponding to the graph; (iii)  $d_i$  (resp.,  $d_j$ ) is the degree of the node  $v_i$  (resp.,  $v_j$ ); (iv)  $\mathcal{C}_i$  (resp.,  $\mathcal{C}_j$ ) is the community to which the node  $v_i$  (resp.,  $v_j$ ) belongs; (v)  $\delta(\mathcal{C}_i, \mathcal{C}_j)$  is the Kronecker delta function, which returns 1 if  $v_i$  and  $v_j$  belong to the same community and 0 otherwise. The greater  $Q$  is, the better the partition of nodes in communities. Our expectation is that the communities extracted from  $G^e(t)$  have associated a higher value of  $Q$  than communities extracted from  $G$  since  $G^e(t)$  “contains” additional knowledge than  $G$ ; such a knowledge is the one provided by  $\mathcal{GNN}$ .

The modularity  $Q$  defined above allows us to analyze the differences in the quality of communities. However,  $Q$  does not allow us to know whether the composition of the communities of  $G$  and  $G^e(t)$  is similar or not. Since  $\mathcal{CS}$  and  $\mathcal{CS}^e(t)$  are sets of communities and each community consists of a set of nodes, the comparison of community compositions is not straightforward. To solve this problem we proceed as follows. For each community  $\mathcal{C}_i \in \mathcal{CS}$ , we compute the Jaccard coefficient [49]  $J_{ij}$  between its nodes and those of each community  $\mathcal{C}_j \in \mathcal{CS}^e(t)$ . Recall that the value of the Jaccard coefficient ranges in the real interval  $[0, 1]$ ; the greater the value of the Jaccard coefficient  $J_{ij}$  of  $\mathcal{C}_i$  and  $\mathcal{C}_j$ , the greater the overlap between these two communities. After computing the Jaccard coefficient between  $\mathcal{C}_i$  and any community  $\mathcal{C}_j \in \mathcal{CS}^e(t)$ , we take the maximum value  $J_i$  of the Jaccard coefficients thus obtained; it represents the maximum possible overlap between  $\mathcal{C}_i$  and a community of  $\mathcal{CS}^e(t)$ . Proceeding in this way, we obtain a value  $J_i$  for each community  $\mathcal{C}_i \in \mathcal{CS}$ . Afterward, we calculate the mean  $J_M$  of all the values  $J_i$  thus obtained.  $J_M$  is an indicator of the structural similarity between the communities of  $\mathcal{CS}$  and  $\mathcal{CS}^e(t)$ . Its value ranges between 0 and 1; the greater  $J_M$ , the greater the structural similarity between the communities of  $G$  and  $G^e(t)$ . The pseudocode describing this behavior is reported in Algorithm 1.

**Algorithm 1** Function COMMUNITY\_STRUCTURAL\_SIMILARITY

**Input**

- $\mathcal{CS}$ : a set of communities of  $G$
- $\mathcal{CS}^e(t)$ : a set of communities of  $G^e(t)$
- $th$ : the minimum size for a community to be considered as such

**Output**

- $J_M$ : the Jaccard coefficient representing the structural similarity between  $\mathcal{CS}$  and  $\mathcal{CS}^e(t)$

```

1: function COMMUNITY_STRUCTURAL_SIMILARITY
2:    $J_M = 0$ 
3:    $n = 0$ 
4:   for  $\mathcal{C}_i$  in  $\mathcal{CS}$  do
5:      $J_i = 0$ 
6:     if  $|\mathcal{C}_i| > th$  then
7:       for  $\mathcal{C}_j$  in  $\mathcal{CS}^e(t)$  do
8:         if  $|\mathcal{C}_j| > th$  then
9:            $J_{ij} = \frac{|\mathcal{C}_i \cap \mathcal{C}_j|}{|\mathcal{C}_i \cup \mathcal{C}_j|}$ 
10:          if  $J_{ij} > J_i$  then
11:             $J_i = J_{ij}$ 
12:          end if
13:        end if
14:      end for
15:       $J_M = J_M + J_i$ 
16:       $n = n + 1$ 
17:    end if
18:  end for
19:  if  $n > 0$  then
20:     $J_M = \frac{J_M}{n}$ 
21:  end if
22:  return  $J_M$ 
23: end function

```

At the end of this analysis, we have two metrics to compare the communities of  $G$  and  $G^e(t)$ , namely the modularity  $Q$  and the average Jaccard coefficient  $J_M$ . In Sect. 4.3, we test whether these measures are useful to train  $\mathcal{GNN}$ , and thus whether they can help improve its performance.

**3.2.2 A measure to compare clustering coefficients in  $G$  and  $G^e(t)$**

Another interesting analysis to compare  $G$  and  $G^e(t)$  is based on clustering coefficients. Recall that the clustering coefficient of a node measures its tendency to cluster with other nodes. In fact, the higher the average clustering coefficient in a graph, the greater the number of closed triads compared to the number of open triads [50]. Since  $G$  and  $G^e(t)$  have weighted arcs, we use the following formula to compute the weighted clustering coefficient  $c_i$  relative to a node  $v_i$  [51]:

$$c_i = \frac{1}{d_i(d_i - 1)} \sum_{v_j \in \text{neigh}(v_i)} \sum_{v_k \in \text{neigh}(v_i), v_k \neq v_j} (\hat{w}_{ij} \hat{w}_{ik} \hat{w}_{jk})^{\frac{1}{3}} \tag{3.3}$$

Here: (i)  $d_i$  is the degree of the node  $v_i$ ; (ii)  $\text{neigh}(v_i)$  represents the neighbors of  $v_i$ , i.e., the nodes connected to  $v_i$  through an edge; (iii)  $\hat{w}_{ij}$  (resp.,  $\hat{w}_{ik}$ ,  $\hat{w}_{jk}$ ) is the normalized weight of the edge  $e_{ij}$  (resp.,  $e_{ik}$ ,  $e_{jk}$ ); if we indicate with  $w_{\max}$  the maximum edge weight in the network, then  $\hat{w}_{ij} = \frac{w_{ij}}{w_{\max}}$  (resp.,  $\hat{w}_{ik} = \frac{w_{ik}}{w_{\max}}$ ,  $\hat{w}_{jk} = \frac{w_{jk}}{w_{\max}}$ ). Following this definition, the higher the weight of a triad is, the higher its importance.

Having the weighted clustering coefficient of each node in the network, we can compute the mean weighted clustering coefficient  $c_M$  of the network as:

$$c_M = \frac{\sum_{v_i \in V} c_i}{|V|} \tag{3.4}$$

The weighted clustering coefficient could be useful to test whether the embeddings of  $\mathcal{GNN}$  tend to increase or decrease the strength of the connections of triads during its

training. We might expect that the edges and structures of  $G^e(t)$  would have different weights than those of  $G$  just because of the embeddings learned from  $\mathcal{GN}\mathcal{N}$ . Similarly to the measures  $Q$  and  $J_M$  seen in the previous section,  $c_M$  might have an impact on the performance of  $\mathcal{GN}\mathcal{N}$ . Therefore, in Sect. 4.4, we present some tests we performed to analyze the differences between the mean weighted clustering coefficients of  $G$  and  $G^e(t)$  and to understand whether  $c_M$  can really support the training of  $G$ .

### 3.2.3 A measure to compare centrality measures in $G$ and $G^e(t)$

Centrality measures represent another interesting viewpoint to investigate for our framework. They indicate the importance of a node in the network in terms of the number of connections, number of shortest paths passing through it, relevance of its neighbors, and so on [52]. Let  $\gamma$  be a centrality measure in a weighted graph. Based on the values of  $\gamma$ , a ranking of the graph nodes can be built. We might think of computing  $\gamma$  on  $G$  and  $G^e(t)$  to obtain the rankings  $R_\gamma$  and  $R_\gamma^e(t)$  of the nodes of  $G$  and  $G^e(t)$  with respect to the values of  $\gamma$ . Following this reasoning, we could verify whether the most important nodes in  $R_\gamma$  are the same or different than the most important nodes in  $R_\gamma^e(t)$ . Such a check would allow us to understand whether  $\mathcal{GN}\mathcal{N}$  has modified the relevance of the nodes of  $G$  based on what it has learned during its training process. Since most centrality measures follow a power law distribution, for each ranking we can focus only on the nodes with the highest values. For instance, we could focus only on the top 20% of the nodes in each ranking.

To check the differences between the two rankings  $R_\gamma$  and  $R_\gamma^e(t)$ , we rely on Kendall's tau coefficient. It is a measure of the correspondence between two rankings. The higher its value, the greater the correspondence between the two rankings into examination [53]. Thus, within our framework, if the value of the Kendall's tau coefficient  $\tau_\gamma$ , calculated on the rankings  $R_\gamma$  and  $R_\gamma^e(t)$ , tends to 1, it means that  $R_\gamma$  and  $R_\gamma^e(t)$  are close, and thus that the embeddings returned by  $\mathcal{GN}\mathcal{N}$  have preserved the importance of the nodes of  $G$  during the training activity. Otherwise, if  $\tau_\gamma$  tends to 0, it means that the rankings are different, and therefore the training of  $\mathcal{GN}\mathcal{N}$  has changed the relevance of the nodes of  $G$ . As a consequence, investigating  $\tau_\gamma$  is interesting to observe whether or not the training of  $\mathcal{GN}\mathcal{N}$  has led to changes in the importance of the nodes in  $G$ . The experiments related to this investigation are explained in detail in Sect. 4.5.

### 3.3 Evaluation and enhancement of a GNN performed by our framework

In the previous sections, we introduced three different perspectives to investigate embeddings learned through a GNN, namely community structure, clustering coefficient and centrality measures. For each of these perspectives, we identified measures allowing us to carry out a quantitative study. These measures are  $Q$  and  $J_M$  for communities,  $c_M$  for clustering coefficient and  $\tau_\gamma$  for centrality measures. Thanks to them, we can study the evolution of the embeddings returned by  $G^e(t)$  during the various epochs of the training of  $\mathcal{GN}\mathcal{N}$ . Such a study could reveal the presence of trends that, in addition to being valuable for the analysis and evaluation of the phenomenon as such, could provide support for improving the performance of  $\mathcal{GN}\mathcal{N}$ . In fact, these measures could add information and knowledge capable of supporting the training of this network. One way for our measures to play a role in improving the performance of  $\mathcal{GN}\mathcal{N}$  is to leverage the loss function  $\mathcal{L}$  used to train this network. In our framework, we do not need to identify a specific loss function but can use any loss function available in the literature, such as binary cross-entropy, categorical cross-entropy and mean squared error-based loss [54].

Therefore, let  $\mathcal{L}$  be the starting loss function that we chose. To add the information that can be derived through our measures, we can introduce a new loss function derived from  $\mathcal{L}$ , which we call  $\mathcal{L}^e$ . It is defined as follows<sup>3</sup>:

$$\mathcal{L}^e = \lambda_1 \mathcal{L} - \lambda_2 Q - \lambda_3 J_M - \lambda_4 c_M - \lambda_5 \tau_\gamma \quad (3.5)$$

Each component of Eq. 3.5 is weighted with a factor  $\lambda_i$  defining its importance. The value of  $\lambda_i$  belongs to the real interval  $[0, 1]$ ; moreover, the sum of all weights is equal to 1, i.e.,  $\sum_{i=1}^5 \lambda_i = 1$ .  $\mathcal{L}^e$  contains both the information carried out by  $\mathcal{L}$  and the one derived through our framework. In defining  $\mathcal{L}^e$  we start from  $\mathcal{L}$  (and, thus, we do not give up its contribution) because  $\mathcal{L}$  is tailored to the Machine Learning task that  $\mathcal{GN}\mathcal{N}$  is supposed to solve, such as the binary cross-entropy for binary classification, the mean squared error for regression, etc. Moreover, we add to it the information carried out by the four measures defined above.

It is worth noting that all the measures we identified were added with a negative sign. Therefore, the greater their value, the smaller the value of  $\mathcal{L}^e$ . In this way, we are forcing  $\mathcal{GN}\mathcal{N}$  to consider the structural information

<sup>3</sup> Actually, as for the measures  $Q$ ,  $J_M$ ,  $c_M$  and  $\tau_\gamma$ , we should have written  $Q^e(t)$ ,  $J_M^e(t)$ ,  $c_M^e(t)$  and  $\tau_\gamma^e(t)$ , since they are associated with the graph  $G^e(t)$ . However, throughout the paper, in order not to burden the notation and since there is no risk of confusion, we decided to use the simplified notation for these measures.



resulting from the current  $G^e(t)$  and the differences between  $G^e(t)$  and  $G$ . Clearly, each measure has its own specific impact in the training of  $\mathcal{GNN}$ . In particular, maximizing  $Q$  implies that  $G^e(t)$  should have the highest possible modularity, which could be extremely important in those tasks requiring communities to be as partitioned as possible. The maximization of  $J_M$  implies that we want to have a good overlap between the communities in  $G$  and  $G^e(t)$ , and therefore that  $\mathcal{GNN}$  should not disrupt the initial network structure. Maximizing  $c_M$  could lead  $G^e(t)$  to have a higher mean weighted clustering coefficient, and thus stronger triads. Finally, maximizing  $\tau_\gamma$  implies that the ranking of nodes with respect to the centrality measure  $\gamma$  should be preserved when moving from  $G$  to  $G^e(t)$ . The details of the proposed loss function and how it changes the training and validation processes of the model  $\mathcal{GNN}$  are provided in Algorithm 2.

**Algorithm 2** Model evaluation and enhancement performed by our framework

After that, it proceeds to calculate the analysis measures  $Q$ ,  $J_M$ ,  $c_M$  and  $\tau_\gamma$  from  $G^e(t)$  and, starting from them and  $\mathcal{L}_{train}$ , calculates  $\mathcal{L}_{train}^e$ . At this point, it begins the validation process during which it first computes  $\mathcal{L}_{val}$  and, then,  $G_{val}^e(t)$  starting from  $G_{val}$  and the node embeddings returned by  $\mathcal{GNN}$ . Afterward, it computes  $Q$ ,  $J_M$ ,  $c_M$ ,  $\tau_\gamma$  and  $\mathcal{L}_{val}^e$ . If  $\mathcal{L}_{val}^e$  is less than the current minimum value of the loss function, it means that, during the current epoch  $t$ ,  $\mathcal{GNN}$  has been improved. In this case, our framework saves the new weights of  $\mathcal{GNN}$  and considers  $\mathcal{L}_{val}^e$  as the new current minimum value of the loss function.

Finally, we would like to point out that the formula of  $\mathcal{L}^e$  specified in Eq. 3.5 is generic, does not depend on the starting loss function adopted and can be applied on any GNN, regardless of the Machine Learning task it must perform. Clearly, it is also possible to use only some, or even one, analysis measures by setting to 0 the weights  $\lambda_i$  corresponding to the measures that we do not want to

---

#### Requires

- $G$ : a starting graph
- $\mathcal{GNN}$ : a Graph Neural Network
- $\mathcal{L}$ : a loss function tailored to the Machine Learning task that  $\mathcal{GNN}$  must perform
- $\eta$ : a positive integer representing the number of training epochs
- $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$ : real numbers in the interval  $[0, 1]$  such that  $\sum_{i=1}^5 \lambda_i = 1$

```

1:  $loss\_val = +\infty$ 
2: Split  $G$  in three subgraphs  $G_{train}, G_{val}, G_{test}$  for training, validation and testing
3: for  $t$  in  $1..\eta$  do
4:   // training
5:   Compute  $\mathcal{L}_{train}$ 
6:   Extract the embeddings of the nodes of  $G_{train}$  obtained by  $\mathcal{GNN}$ 
7:   Use these embeddings to compute  $G_{train}^e(t)$  from  $G_{train}$ 
8:   Compute  $Q, J_M, c_M, \tau_\gamma$  from  $G_{train}^e(t)$ 
9:    $\mathcal{L}_{train}^e = \lambda_1 \mathcal{L}_{train} - \lambda_2 Q - \lambda_3 J_M - \lambda_4 c_M - \lambda_5 \tau_\gamma$ 
10:  Compute the gradients of  $\mathcal{L}_{train}^e$  and update the weights of  $\mathcal{GNN}$ 
11:  // validation
12:  Compute  $\mathcal{L}_{val}$ 
13:  Extract the embeddings of the nodes of  $G_{val}$  obtained by  $\mathcal{GNN}$ 
14:  Use these embeddings to compute  $G_{val}^e(t)$  from  $G_{val}$ 
15:  Compute  $Q, J_M, c_M, \tau_\gamma$  from  $G_{val}^e(t)$ 
16:   $\mathcal{L}_{val}^e = \lambda_1 \mathcal{L}_{val} - \lambda_2 Q - \lambda_3 J_M - \lambda_4 c_M - \lambda_5 \tau_\gamma$ 
17:  if  $\mathcal{L}_{val}^e < loss\_val$  then
18:    Store the weights of  $\mathcal{GNN}$  as the current best weights of it
19:     $loss\_val = \mathcal{L}_{val}^e$ 
20:  end if
21: end for

```

---

As reported in this algorithm, after splitting  $G$  in three subgraphs  $G_{train}$ ,  $G_{val}$  and  $G_{test}$ , our framework trains  $\mathcal{GNN}$  for  $\eta$  epochs. During the current epoch  $t$ , it calculates  $\mathcal{L}_{train}$  by applying the loss function chosen for the task performed by the  $\mathcal{GNN}$ . Then, it applies  $\mathcal{GNN}$  on  $G_{train}$  to compute node embeddings. Afterward, it computes  $G_{train}^e(t)$  starting from  $G_{train}$  and node embeddings computed previously.

employ. Indeed, starting from  $\mathcal{L}^e$ , by setting some of its weights to 0, it is possible to obtain various specific functions, which can be used to study the contribution of each measure on the scenario under consideration.

In particular, by setting  $\lambda_3 = \lambda_4 = \lambda_5 = 0$ , we obtain a particular version of  $\mathcal{L}^e$ , which we call  $\mathcal{L}_Q^e$ , defined as:

**Table 1** Some statistics of the adopted datasets

	Cora	Chameleon	Actor	Cornell	Texas	Wisconsin
Number of nodes	2708	2277	7600	183	183	251
Number of edges	5278	31,371	26,659	277	279	450
Density	0.014	0.012	0.001	0.017	0.017	0.014
Clustering coefficient	0.241	0.208	0.080	0.167	0.198	0.208

$$\mathcal{L}_Q^e = \lambda_1 \mathcal{L} - \lambda_2 Q \tag{3.6}$$

Now, since the sum of the weights of  $\mathcal{L}^e$  must be equal to 1, we have that  $\lambda_2 = 1 - \lambda_1$ . Furthermore, by setting  $\lambda = 1 - \lambda_1$ , we have that  $\mathcal{L}_Q^e$  can be defined as:

$$\mathcal{L}_Q^e = (1 - \lambda) \mathcal{L} - \lambda Q \tag{3.7}$$

This specialization of  $\mathcal{L}^e$  emphasizes the role of modularity in the training of a GNN. Therefore, it can be very useful when we want to analyze this role. In fact, we employ it in Sect. 4.3, devoted to this task.

Instead, by setting  $\lambda_2 = \lambda_4 = \lambda_5 = 0$  and proceeding similarly to what we have seen for  $\mathcal{L}_Q^e$ , we obtain a particular version of  $\mathcal{L}^e$ , which we call  $\mathcal{L}_J^e$ , defined as:

$$\mathcal{L}_J^e = (1 - \lambda) \mathcal{L} - \lambda J_M \tag{3.8}$$

$\mathcal{L}_J^e$  emphasizes the role of the Jaccard coefficient  $J_M$  in the training of a GNN. We employ it in Sect. 4.3, where this role is studied in detail.

Similarly, by setting  $\lambda_2 = \lambda_3 = \lambda_5 = 0$  and performing the same operations as in the previous two cases, we obtain a particular version of  $\mathcal{L}^e$ , which we call  $\mathcal{L}_{c_M}^e$ , defined as:

$$\mathcal{L}_{c_M}^e = (1 - \lambda) \mathcal{L} - \lambda c_M \tag{3.9}$$

$\mathcal{L}_{c_M}^e$  emphasizes the role of the clustering coefficient on the training of a GNN. We adopted it in Sect. 4.4, devoted to study this role.

We end this presentation of the specializations of  $\mathcal{L}^e$  (albeit many more could be defined by appropriately setting the weights  $\lambda_i$ ) by setting  $\lambda_2 = \lambda_3 = \lambda_4 = 0$  and proceeding as in the previous cases. We obtain a particular version of  $\mathcal{L}^e$ , which we call  $\mathcal{L}_\gamma^e$ , defined as:

$$\mathcal{L}_\gamma^e = (1 - \lambda) \mathcal{L} - \lambda \tau_\gamma \tag{3.10}$$

$\mathcal{L}_\gamma^e$  emphasizes the role of the centrality measure  $\gamma$  in the training of a GNN. It will be employed in Sect. 4.5 where

we investigate the role of the degree centrality in the training of a GNN.

In Eq. 3.7 (resp., 3.8, 3.9, 3.10), the higher the weight assigned to  $Q$  (resp.,  $J_M, c_M, \tau_\gamma$ ) and the lower the weight assigned to  $\mathcal{L}$ . Clearly, when  $\lambda = 0$ , the GNN is trained only with  $\mathcal{L}$ , while when  $\lambda = 1$ , the information carried by  $Q$  (resp.,  $J_M, c_M, \tau_\gamma$ ) is the only one that contributes to the training of the GNN.

## 4 Experiments

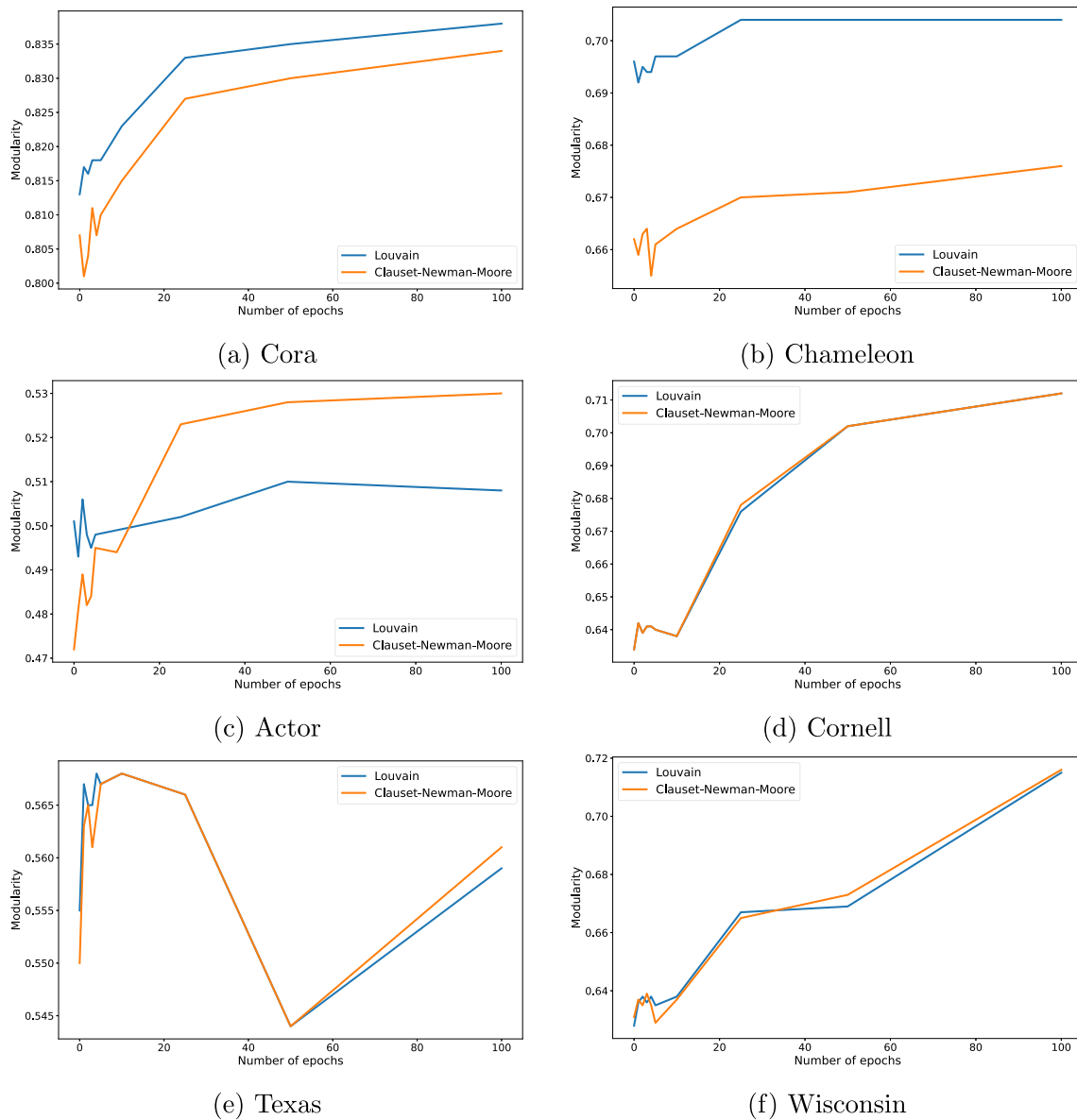
In this section, we present the experiments we conducted to test our framework. Specifically, in Sect. 4.1, we provide an overview of the datasets used. In Sect. 4.2, we describe the GNN employed. In Sects. 4.3, 4.4 and 4.5, we illustrate the experiments regarding community structures, clustering coefficient and centrality measures, respectively. Finally, in Sect. 4.6, we evaluate the ability of our framework to enhance the training of the underlying GNN thanks to the information extracted through it.

### 4.1 Datasets

To ensure a robust validation of our framework, in our experiments we employed six different datasets widely adopted in the fields of GNN and Network Analysis. By leveraging these datasets we could thoroughly evaluate the performance and effectiveness of our framework under various conditions and scenarios. The first dataset is Cora [55]; it consists of a citation network in which each node represents a scholarly paper and an arc connecting two nodes indicates that one paper cites the other. The second dataset is Chameleon [56], which contains page-page networks focusing on particular topics, such as chameleons. In this dataset, nodes correspond to articles while edges indicate reciprocal links between articles. The third dataset

**Table 2** Hyperparameters of FSGNN for each dataset

	Cora	Chameleon	Actor	Cornell	Texas	Wisconsin
$WD_{fc1}$	0.001	0.0	0.001	0.001	0.001	0.001
$WD_{fc2}$	0.0001	0.0	0.0001	0.001	0.0	0.0001
$LR_{fc}$	0.01	0.005	0.01	0.01	0.01	0.01
Dropout	0.25	0.25	0.25	0.25	0.25	0.25



**Fig. 1** Modularity  $Q$  of  $G^e(t)$  against training epochs for the datasets of interest

is Actor [57]; it is a network of actor co-occurrences; in it, each node represents an actor while an edge between two nodes denotes the co-occurrence of the corresponding actors on the same Wikipedia page. The fourth, fifth, and sixth datasets are Cornell, Texas and Wisconsin [58], which are all of the same type with nodes representing web pages and arcs denoting hyperlinks from one page to another. The statistics of the six datasets used are shown in Table 1.

For the sake of space, we decided to work only with one Machine Learning task typically performed by GNNs, namely node classification. However, we point out that our framework can handle any Machine Learning task as long as node embeddings can be extracted through it.

### 4.2 Reference GNN model

The GNN model we adopted for our experiments is the Feature Selection Graph Neural Network (FSGNN) introduced in [59]. We decided to use this model because of its high performance in the node classification task, which makes it a good baseline to evaluate the effectiveness of our framework. FSGNN is a two-layered GNN model designed to handle node classification tasks. Because of the significant differences between the datasets of interest (see Table 1), we had to provide different values to its hyperparameters, depending on the dataset on which it was applied. The hyperparameters of FSGNN are: (i) the decay of the weights for the first and second fully connected

**Table 3** Average values of classification metrics obtained by FSGNN using  $Q$  in the training loss function—Cora, Chameleon, and Actor datasets

$\lambda$	Cora			Chameleon			Actor		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
0.0	0.858	0.867	0.823	0.733	<b>0.737</b>	0.736	0.342	0.276	0.300
0.1	<b>0.863</b>	<b>0.875</b>	<b>0.830</b>	<b>0.736</b>	<b>0.737</b>	<b>0.737</b>	0.340	0.329	0.298
0.2	0.858	0.871	0.820	0.732	0.735	0.735	0.338	0.305	0.296
0.3	0.858	0.869	0.824	0.732	<b>0.737</b>	0.734	0.343	0.296	0.302
0.4	0.854	0.873	0.813	0.732	<b>0.737</b>	0.734	0.344	0.318	0.302
0.5	0.858	0.870	0.818	0.725	0.727	0.726	0.344	0.297	0.308
0.6	0.851	0.871	0.810	0.731	0.734	0.733	0.344	0.327	0.307
0.7	0.856	0.874	0.814	0.722	0.722	0.723	<b>0.353</b>	0.325	0.322
0.8	0.851	0.866	0.810	0.695	0.702	0.695	0.351	0.335	0.324
0.9	0.843	0.838	0.784	0.661	0.671	0.659	<b>0.353</b>	<b>0.341</b>	<b>0.326</b>
1.0	0.231	0.034	0.140	0.184	0.037	0.200	0.224	0.049	0.200

The values in bold are the highest ones

**Table 4** Average values of classification metrics obtained by FSGNN using  $Q$  in the training loss function—Cornell, Texas and Wisconsin datasets

$\lambda$	Cornell			Texas			Wisconsin		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
0.0	0.757	0.560	0.496	0.758	0.518	0.447	0.836	0.635	0.594
0.1	0.762	0.566	0.495	0.768	0.518	0.460	0.835	0.672	0.593
0.2	0.757	0.561	0.496	0.768	0.529	0.456	0.831	0.625	0.580
0.3	0.789	0.592	0.526	0.768	0.516	0.460	0.831	0.630	0.591
0.4	0.768	0.540	0.493	0.778	0.563	0.474	0.824	0.627	0.582
0.5	0.762	0.536	0.471	0.795	0.578	0.498	0.839	0.641	0.591
0.6	0.768	0.569	0.501	0.805	0.583	0.505	0.839	0.636	0.590
0.7	0.784	0.574	0.518	0.800	0.503	0.499	0.851	0.607	0.575
0.8	0.805	<b>0.598</b>	0.532	<b>0.849</b>	<b>0.613</b>	<b>0.556</b>	<b>0.867</b>	<b>0.698</b>	<b>0.626</b>
0.9	<b>0.811</b>	0.569	<b>0.543</b>	0.816	0.532	0.511	0.859	0.651	0.618
1.0	0.205	0.054	0.224	0.189	0.038	0.200	0.169	0.046	0.210

The values in bold are the highest ones

**Table 5** Average values of classification metrics obtained by FSGNN using  $J_M$  in the training loss function—Cora, Chameleon, and Actor datasets

$\lambda$	Cora			Chameleon			Actor		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
0.0	0.858	0.867	0.823	0.733	0.737	0.736	<b>0.342</b>	0.276	0.300
0.1	0.856	0.869	0.822	0.734	0.736	0.735	0.341	<b>0.322</b>	0.299
0.2	<b>0.862</b>	0.875	<b>0.825</b>	<b>0.739</b>	<b>0.740</b>	<b>0.741</b>	0.341	0.265	0.297
0.3	0.861	0.877	0.820	0.730	0.733	0.731	0.341	0.268	<b>0.301</b>
0.4	0.855	0.874	0.816	0.734	0.738	0.736	0.326	0.223	0.282
0.5	0.860	<b>0.879</b>	0.816	0.724	0.724	0.725	0.251	0.173	0.202
0.6	0.858	0.877	0.810	0.723	0.730	0.725	0.252	0.158	0.202
0.7	0.861	0.876	0.816	0.716	0.725	0.715	0.246	0.094	0.200
0.8	0.851	0.844	0.794	0.399	0.309	0.403	0.246	0.049	0.200
0.9	0.847	0.846	0.788	0.203	0.041	0.200	0.246	0.049	0.200
1.0	0.186	0.05	0.141	0.184	0.037	0.200	0.215	0.043	0.200

The values in bold are the highest ones

**Table 6** Average values of classification metrics obtained by FSGNN using  $J_M$  in the training loss function—Cornell, Texas and Wisconsin datasets

$\lambda$	Cornell			Texas			Wisconsin		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
0.0	0.757	0.560	0.496	0.758	0.518	0.447	0.836	<b>0.635</b>	<b>0.594</b>
0.1	0.762	0.561	0.495	0.762	0.517	0.455	0.827	0.622	0.581
0.2	0.768	0.568	0.494	0.773	0.537	0.466	0.824	0.622	0.570
0.3	0.800	0.594	0.536	0.768	0.517	0.463	0.827	0.625	0.583
0.4	0.773	0.543	0.495	0.789	0.578	0.484	0.827	0.627	0.584
0.5	0.773	0.560	0.495	0.795	0.576	0.498	0.824	0.623	0.572
0.6	0.784	0.598	0.516	0.800	0.517	0.490	0.839	0.600	0.567
0.7	0.805	<b>0.616</b>	<b>0.560</b>	0.811	0.546	0.522	0.839	0.604	0.562
0.8	<b>0.816</b>	<b>0.616</b>	<b>0.560</b>	<b>0.843</b>	<b>0.607</b>	<b>0.564</b>	<b>0.855</b>	0.624	0.574
0.9	0.800	0.560	0.535	0.822	0.529	0.523	0.839	0.578	0.558
1.0	0.211	0.042	0.200	0.189	0.038	0.200	0.176	0.035	0.200

The values in bold are the highest ones

layers ( $WD_{fc1}$  and  $WD_{fc2}$ ), (ii) the learning rate of the fully connected layers ( $LR_{fc}$ ), and (iii) the dropout rate. The corresponding values are reported in Table 2.

### 4.3 Experiments on communities structures

In this section, we describe the experiments performed to analyze the communities in  $G$  and  $G^e(t)$  for the datasets selected.

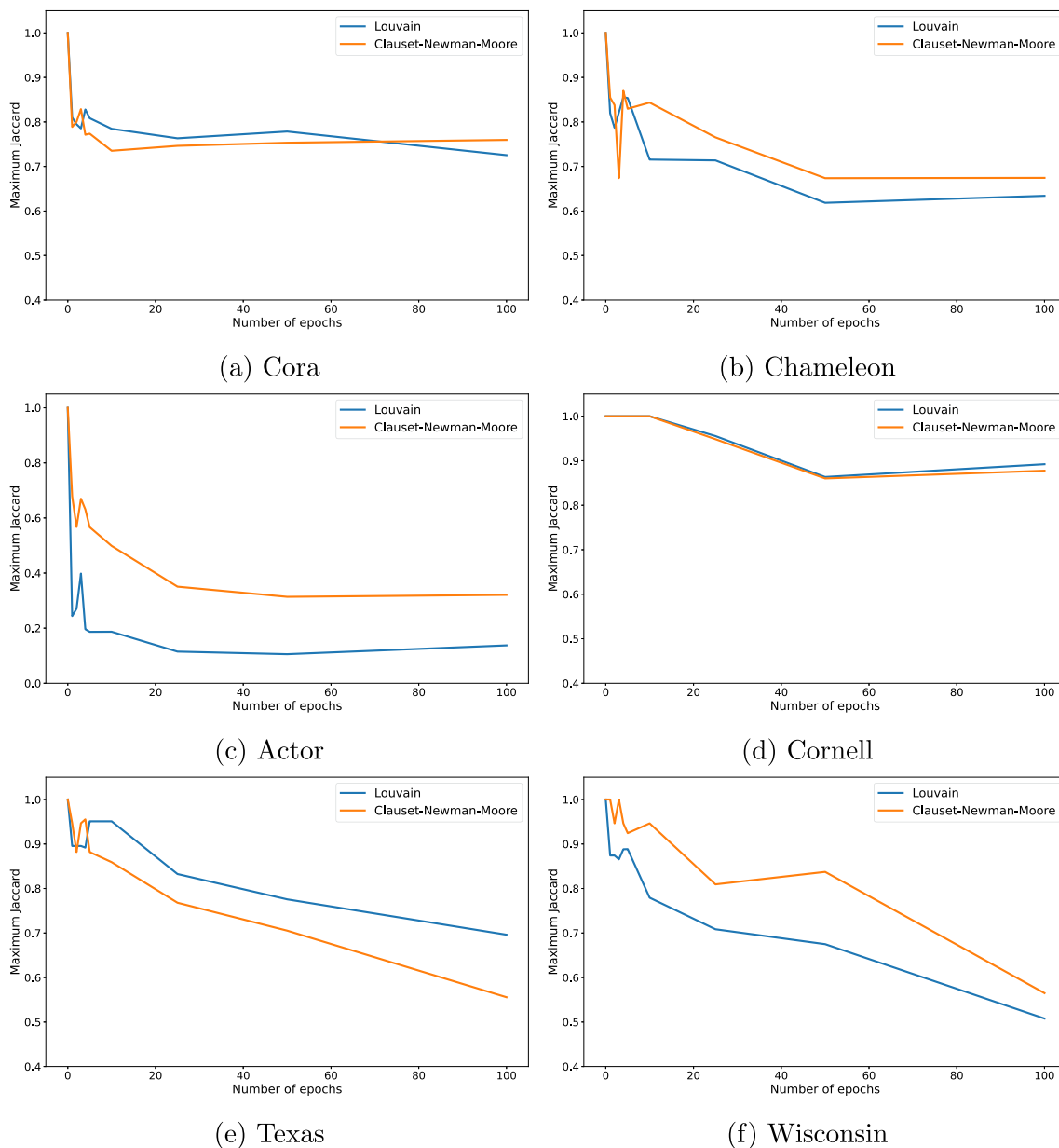
First, we tested whether the modularity  $Q$  of  $G^e(t)$  changed during the training epochs, which could show that FSGNN was creating stronger or weaker communities during its learning. To this end, we trained FSGNN for 100 epochs. At the end of each epoch, we extracted the communities from  $G^e(t)$  through the Louvain and Clauset-Newman-Moore algorithms and, after that, calculated the modularity of  $G^e(t)$ . We used two different community extraction algorithms to better verify the stability of the results obtained. The corresponding results for each dataset are shown in Fig. 1.

From the analysis of this figure, we can observe that in Cora, Chameleon, Actor, Cornell and Wisconsin the modularity of  $G^e(t)$  increased as the number of epochs increased. This means that FSGNN created more cohesive communities than those in the original graph. Specifically, in this case, the increase in  $Q$  ranges from 1.8%, obtained for the Texas dataset, to 12.70%, obtained for the Cornell dataset. Only in the Texas dataset we do not observe a stable growth of  $Q$ , due to some fluctuations. However, even in this less favorable case, the modularity of  $G^e(t)$  at the end of training is greater than that at the beginning of this task. This confirms our hypothesis that the modularity grows as the number of epochs increases.

To test whether modularity can improve the performance of FSGNN, we added the information associated

with  $Q$  into the model training. For this purpose, we adopted the specialization of  $\mathcal{L}^e$  called  $\mathcal{L}_Q^e$  defined in Eq. 3.7. We repeated the experiments five times with different training and testing splits. In Tables 3 and 4, we report the average values of the classification performance metrics we obtained. In these tables, as well as in the next ones, we use bold to indicate the maximum values in the corresponding column. For these tables, as well as for Tables 5 and 6 below, we report only the results obtained with the communities returned by the Louvain algorithm. However, the results obtained with the communities returned by the Clauset-Newman-Moore algorithm are similar.

From the analysis of these tables, we can observe that the use of  $Q$  in the training loss function of FSGNN increases the classification performance of this model. In the Cora and Chameleon datasets, the best results are obtained for  $\lambda = 0.1$ ; in this case, the values of classification metrics increase only slightly. Instead, in the Actor dataset we achieve interesting performances when  $\lambda = 0.9$ , in which case Accuracy increases by 3.2%, Precision by 23.6% and Recall by 8.7%. Similarly, in the Cornell, Texas and Wisconsin datasets, the performance gets interesting results when  $\lambda = 0.8$  or  $\lambda = 0.9$ . More specifically, for the Cornell dataset, Accuracy increases by 7.1%, Precision by 6.7% and Recall by 9.5%. As for the Texas dataset, Accuracy increases by 10.7%, Precision by 15.5% and Recall by 19.6%. Finally, for what concerns the Wisconsin dataset, Accuracy increases by 3.3%, Precision by 9.9% and Recall by 4.7%. Hence, we can conclude that the adoption of  $G^e(t)$  and  $Q$  during the training of FSGNN results in significant improvements in classification performance in four datasets (i.e., Actor, Cornell, Texas, and Wisconsin), while it leads to marginal improvements in the remaining two ones (i.e., Cora and Chameleon). Still for

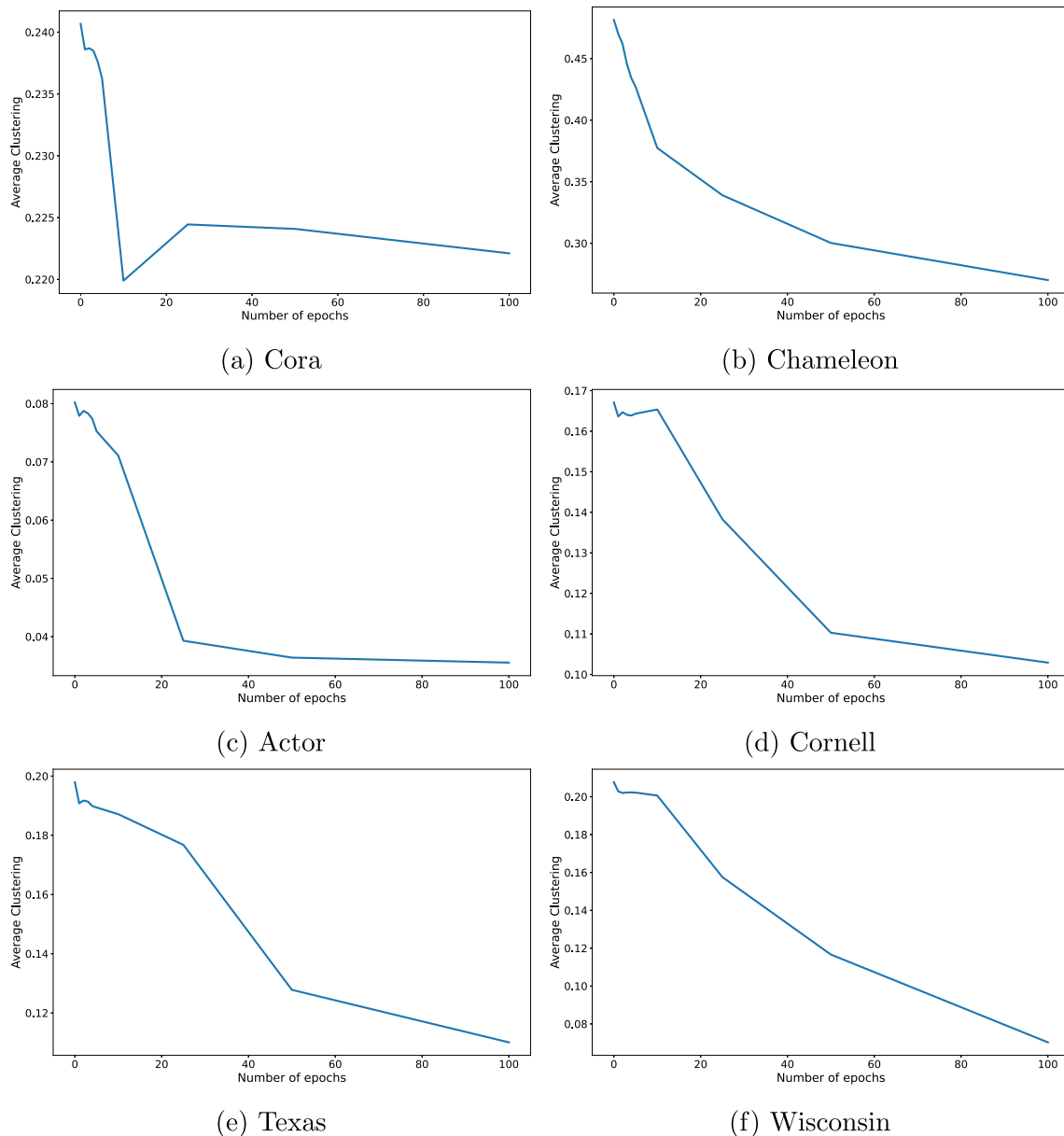


**Fig. 2** Jaccard coefficient  $J_M$  of  $G^e(t)$  against training epochs for the datasets of interest

the community investigation task, we next tested the role of the Jaccard coefficient  $J_M$  to see whether and to what extent the overlapping level between  $G$  and  $G^e(t)$  varies against training epochs. Again, we trained FSGNN for 100 epochs. At the end of each epoch, we extracted communities from  $G$  and  $G^e(t)$  using the Louvain and Clauset-Newman-Moore algorithms. Afterward, we calculated  $J_M$  by applying the procedure described in Algorithm 1. In Fig. 2, we show the values of  $J_M$  during the training of FSGNN.

From the analysis of this figure, we can see that  $J_M$  tends to decrease as the number of epochs increases. In fact, at epoch 0, when training has not yet begun,  $J_M$  is equal to 1, which implies that the communities in  $G$  and  $G^e(t)$  are the

same. Then  $J_M$  decreases as the number of epochs increases and, at the end of the 100th epoch, it reaches a value that is always less than its original one. This occurs regardless of which algorithm we use to detect communities. In particular, for the Cora and Chameleon datasets,  $J_M$  decreases rapidly and reaches a value close to 0.65, which means that there is 65% overlapping between the communities of  $G$  and  $G^e(t)$ . A similar reasoning applies to the Texas and Wisconsin datasets, for which  $J_M$  decreases rapidly and reaches a value close to 0.60. As for the Actor dataset,  $J_M$  decreases very rapidly and settles at 0.40 in the case of adoption of the Clauset-Newman-Moore algorithm, while it settles to 0.10 in the case of employment of the Louvain



**Fig. 3** Average weighted clustering coefficient  $c_M$  of  $G^e(t)$  against training epochs for the datasets of interest

algorithm. Finally, for the Cornell dataset,  $J_M$  fluctuates in a range between 0.86 and 0.92.

Next, we tested whether  $J_M$  could improve the performance of FSGNN in classifying nodes. To this end, we added the information brought by  $J_M$  into the training process of FSGNN. To do this, we employed the specialization  $\mathcal{L}_J^e$  of the function  $\mathcal{L}^e$  defined in Eq. 3.8. We repeated the training of FSGNN five times with different training and testing splits. We report the average values of the classification metrics in Tables 5 and 6.

From the analysis of these tables, we can see that, in many cases,  $J_M$  improves the performance of FSGNN although there are differences among the datasets. In fact,

in the Cora and Chameleon datasets, we observe slight improvements in Accuracy, Precision, and Recall. In the Actor dataset, Precision increases by 16.7% when  $\lambda = 0.1$ , while Accuracy and Recall do not show significant changes. Furthermore, in the Cornell dataset, we get an increase in Accuracy of 7.8%, an increase in Precision of 10.0%, and an increase in Recall of 12.9% for  $\lambda = 0.8$ . Similarly, in the Texas dataset, we obtained a significant increase. In contrast, in the Wisconsin dataset, we can see that Accuracy increased by 2.4%, while Precision and Recall decreased by 1.7% and 3.2%, respectively.

In conclusion, we observe that the knowledge on community structures extracted during the training of FSGNN

**Table 7** Average values of classification metrics obtained by FSGNN using  $c_M$  in the training loss function—Cora, Chameleon, and Actor datasets

$\lambda$	Cora			Chameleon			Actor		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
0.0	0.858	0.867	0.823	0.733	<b>0.737</b>	0.736	0.342	0.276	0.300
0.1	<b>0.863</b>	<b>0.875</b>	<b>0.830</b>	<b>0.736</b>	<b>0.737</b>	<b>0.737</b>	0.340	0.315	0.298
0.2	0.858	0.871	0.820	0.732	0.735	0.735	0.341	0.271	0.298
0.3	0.858	0.869	0.824	0.732	<b>0.737</b>	0.734	0.346	0.277	0.303
0.4	0.854	0.873	0.813	0.732	<b>0.737</b>	0.734	0.346	<b>0.319</b>	<b>0.305</b>
0.5	0.858	0.870	0.818	0.725	0.727	0.726	0.344	0.277	0.303
0.6	0.851	0.871	0.810	0.731	0.734	0.733	0.344	0.311	0.303
0.7	0.856	0.874	0.814	0.722	0.722	0.723	0.345	0.264	0.302
0.8	0.851	0.866	0.810	0.695	0.702	0.695	0.346	0.278	<b>0.305</b>
0.9	0.843	0.838	0.784	0.661	0.671	0.659	<b>0.347</b>	0.270	0.299
1.0	0.231	0.034	0.140	0.184	0.037	0.200	0.225	0.057	0.197

The values in bold are the highest ones

**Table 8** Average values of classification metrics obtained by FSGNN using  $c_M$  in the training loss function—Cornell, Texas and Wisconsin datasets

$\lambda$	Cornell			Texas			Wisconsin		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
0.0	0.757	0.560	0.496	0.758	0.518	0.447	0.836	0.635	0.594
0.1	0.762	0.566	0.495	0.768	0.518	0.460	0.835	0.672	0.593
0.2	0.757	0.561	0.496	0.768	0.529	0.456	0.831	0.625	0.580
0.3	0.789	0.592	0.526	0.768	0.516	0.460	0.831	0.630	0.591
0.4	0.768	0.540	0.493	0.778	0.563	0.474	0.824	0.627	0.582
0.5	0.762	0.536	0.471	0.795	0.578	0.498	0.839	0.641	0.591
0.6	0.768	0.569	0.501	0.805	0.583	0.505	0.839	0.636	0.590
0.7	0.784	0.574	0.518	0.800	0.503	0.499	0.851	0.607	0.575
0.8	0.805	<b>0.598</b>	0.532	<b>0.849</b>	<b>0.613</b>	<b>0.556</b>	<b>0.867</b>	<b>0.698</b>	<b>0.626</b>
0.9	<b>0.811</b>	0.569	<b>0.543</b>	0.816	0.532	0.511	0.859	0.651	0.618
1.0	0.205	0.054	0.224	0.189	0.038	0.200	0.169	0.046	0.210

The values in bold are the highest ones

can be used to improve its performance. For this purpose, it is possible to employ the two analysis measures  $Q$  and  $J_M$  that summarize the knowledge on the variations of community structures extracted during the training of FSGNN.

#### 4.4 Experiments on clustering coefficient

The first experiment on the role of the clustering coefficient in the training of FSGNN aimed to analyze the evolution of the values of the mean weighted clustering coefficient  $c_M$  in  $G^e(t)$  after each training epoch. Specifically, we trained FSGNN for 100 epochs and, at the end of each epoch, we created  $G^e(t)$ , calculated the weighted clustering coefficients for all nodes, and finally computed  $c_M$ . The results obtained are shown in Fig. 3.

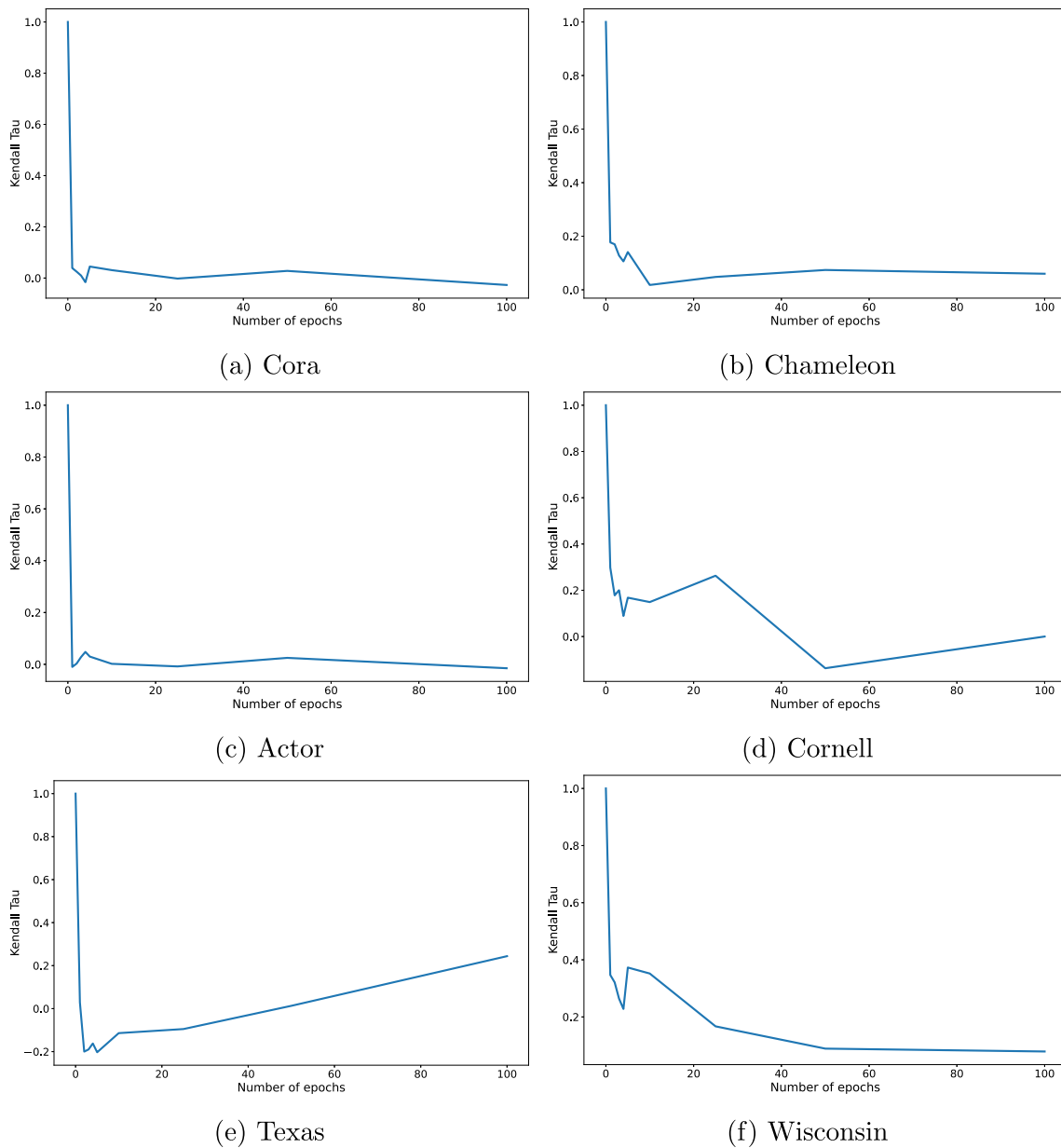
From the analysis of this figure we can observe that, in all cases, the value of  $c_M$  decreases as the number of epochs increases. This means that the triads present in  $G$

lose power during the training of FSGNN. In particular, we observe a decrease in  $c_M$  ranging from 6.2% in the Cora dataset to 62.5% in the Wisconsin dataset. This is an interesting result in that it tells us that the training of FSGNN leads to a decrease in the power exerted by the weighted triads of  $G$ .

To check whether and how much  $c_M$  exerts an influence in the training of FSGNN, we proceeded employing the specialization  $\mathcal{L}_{c_M}^e$  of the function  $\mathcal{L}^e$ , defined in Eq. 3.9. Afterward, we tested  $\mathcal{L}_{c_M}^e$  for different values of  $\lambda$ . Again, we repeated this experiment five times with different training and testing splits. In Tables 7 and 8, we report the results obtained.

From the analysis of these tables, we can observe trends similar to those we have seen for community structures. In particular, we note that the benefits of using  $c_M$  in the Cora and Chameleon datasets are small, while they become significant for the Actor, Cornell, Texas and Wisconsin





**Fig. 4** Kendall’s tau coefficient  $\tau_{wd}$ , computed on the Weighted Degree Centrality, against training epochs for the datasets of interest

datasets. In fact, as for the Actor dataset, we achieved an increase in Precision of 15.6%, in Accuracy of 1.5% and in Recall of 1.7%. As for the Cornell dataset, we obtained an increase in Accuracy of 7.1%, in Precision of 6.8% and in Recall of 9.5%. In the case of Texas, we achieved a higher Accuracy of 10.7%, a higher Precision of 15.7% and a higher Recall of 19.6%. Finally, in the Wisconsin dataset, Accuracy increased by 3.3%, Precision by 9.9% and Recall by 4.7%.

The results obtained lead us to conclude that the mean weighted clustering coefficient  $c_M$  of  $G^e(t)$  changes against training epochs. Therefore, it could be a relevant factor to be investigated during the learning of a GNN model. As

evidence of this, the introduction of  $c_M$  in the loss function allowed us to obtain higher values of the performance metrics than those of the baseline.

### 4.5 Experiments on centrality measures

Our experiment on centrality measures involved the analysis of the trend of the Kendall’s tau coefficient  $\tau_\gamma$  against training epochs. Recall that  $\tau_\gamma$  is an indicator of the agreement of the rankings of the main nodes of  $G$  and  $G^e(t)$  with respect to the centrality measure  $\gamma$ . The first decision we had to make was the choice of  $\gamma$  and of the way to select the main nodes. Regarding centrality measures, we know

**Table 9** Average values of classification metrics obtained by FSGNN using  $\tau_{wd}$  in the training loss function—Cora, Chameleon, and Actor datasets

$\lambda$	Cora			Chameleon			Actor		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
0.0	0.858	0.867	0.823	0.733	<b>0.737</b>	0.736	0.342	0.276	0.300
0.1	0.857	0.868	0.824	<b>0.736</b>	<b>0.737</b>	<b>0.737</b>	0.340	0.315	0.298
0.2	0.859	0.871	<b>0.825</b>	0.732	0.735	0.735	0.339	0.269	0.296
0.3	0.859	0.873	0.821	0.731	0.735	0.732	0.346	0.277	0.303
0.4	0.855	0.876	0.814	0.732	0.736	0.734	0.346	<b>0.319</b>	0.305
0.5	0.857	0.874	0.816	0.729	0.730	0.731	0.344	0.277	0.303
0.6	0.859	0.875	0.818	0.727	0.731	0.729	0.345	0.315	0.304
0.7	<b>0.862</b>	<b>0.880</b>	0.823	0.722	0.725	0.724	0.345	0.318	0.302
0.8	0.858	0.876	0.819	0.714	0.716	0.715	0.345	0.278	0.304
0.9	0.857	0.878	0.812	0.689	0.694	0.689	<b>0.349</b>	0.278	<b>0.307</b>
1.0	0.186	0.050	0.141	0.182	0.043	0.197	0.230	0.080	0.200

The values in bold are the highest ones

**Table 10** Average values of classification metrics obtained by FSGNN using  $\tau_{wd}$  in the training loss function—Cornell, Texas and Wisconsin datasets

$\lambda$	Cornell			Texas			Wisconsin		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
0.0	0.757	0.560	0.496	0.758	0.518	0.447	0.836	0.635	0.594
0.1	0.762	0.561	0.495	0.762	0.517	0.455	0.831	0.623	0.583
0.2	0.768	0.569	0.494	0.773	0.537	0.466	0.831	0.625	0.580
0.3	0.800	0.594	0.536	0.768	0.517	0.463	0.827	0.626	0.589
0.4	0.773	0.543	0.495	0.789	0.578	0.484	0.827	0.627	0.584
0.5	0.773	0.560	0.495	0.795	0.576	0.498	0.824	0.623	0.578
0.6	0.773	0.594	0.512	0.800	0.517	0.49	0.835	0.634	0.595
0.7	0.805	<b>0.616</b>	<b>0.560</b>	0.811	0.546	0.522	0.855	0.649	0.615
0.8	<b>0.816</b>	<b>0.616</b>	<b>0.560</b>	<b>0.843</b>	<b>0.607</b>	<b>0.564</b>	0.855	<b>0.686</b>	0.621
0.9	0.800	0.560	0.535	0.822	0.529	0.523	<b>0.867</b>	0.646	<b>0.629</b>
1.0	0.211	0.042	0.200	0.189	0.038	0.200	0.176	0.035	0.200

The values in bold are the highest ones

**Table 11** Average values of classification metrics obtained by FSGNN using different combinations of  $\mathcal{L}^e$  as training loss function—Cora, Chameleon and Actor datasets

$\lambda$	Cora			Chameleon			Actor		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Baseline	<b>0.858</b>	0.867	<b>0.823</b>	<b>0.733</b>	<b>0.737</b>	<b>0.736</b>	<b>0.342</b>	0.276	0.300
$\lambda_1, \lambda_2, \lambda_3$	0.851	0.868	0.812	0.702	0.704	0.703	<b>0.342</b>	0.281	0.301
$\lambda_1, \lambda_2, \lambda_4$	0.852	0.868	0.811	0.719	0.722	0.721	0.340	0.286	0.299
$\lambda_1, \lambda_2, \lambda_5$	0.850	<b>0.877</b>	0.806	0.729	0.732	0.730	0.338	0.258	0.293
$\lambda_1, \lambda_3, \lambda_4$	0.854	0.872	0.813	0.706	0.707	0.707	<b>0.342</b>	0.281	0.301
$\lambda_1, \lambda_3, \lambda_5$	0.850	<b>0.877</b>	0.806	0.728	0.729	0.728	0.309	0.178	0.264
$\lambda_1, \lambda_4, \lambda_5$	0.850	<b>0.877</b>	0.806	0.722	0.725	0.722	0.332	0.229	0.287
$\lambda_1, \lambda_2, \lambda_3, \lambda_4$	0.853	0.867	0.812	0.684	0.696	0.686	<b>0.342</b>	0.281	0.301
$\lambda_1, \lambda_2, \lambda_3, \lambda_5$	0.856	0.875	0.813	0.695	0.700	0.696	0.340	0.313	<b>0.304</b>
$\lambda_1, \lambda_2, \lambda_4, \lambda_5$	0.851	0.873	0.807	0.694	0.700	0.696	<b>0.342</b>	0.291	0.303
$\lambda_1, \lambda_3, \lambda_4, \lambda_5$	0.856	0.875	0.813	0.702	0.709	0.704	0.338	0.258	0.293
$\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$	0.849	0.869	0.803	0.692	0.700	0.691	0.340	<b>0.314</b>	0.302

The values in bold are the highest ones

**Table 12** Average values of classification metrics obtained by FSGNN using different combinations of  $\mathcal{L}^e$  as training loss function—Cornell, Texas and Wisconsin datasets

$\lambda$	Cornell			Texas			Wisconsin		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Baseline	0.757	0.560	0.496	0.758	0.518	0.447	0.836	0.635	0.594
$\lambda_1, \lambda_2, \lambda_3$	0.784	0.576	0.513	0.789	<b>0.574</b>	0.488	0.827	0.629	0.584
$\lambda_1, \lambda_2, \lambda_4$	0.762	0.567	0.494	0.789	<b>0.574</b>	0.488	0.831	0.630	0.587
$\lambda_1, \lambda_2, \lambda_5$	0.757	0.557	0.48	0.784	0.554	0.476	0.831	0.630	0.587
$\lambda_1, \lambda_3, \lambda_4$	0.762	0.555	0.482	0.789	<b>0.574</b>	0.488	0.831	0.629	0.587
$\lambda_1, \lambda_3, \lambda_5$	0.751	0.537	0.467	0.784	0.554	0.476	0.827	0.629	0.584
$\lambda_1, \lambda_4, \lambda_5$	0.757	0.553	0.476	0.784	0.554	0.476	0.824	0.585	0.569
$\lambda_1, \lambda_2, \lambda_3, \lambda_4$	0.789	0.594	0.533	0.805	0.548	0.494	0.851	0.647	0.617
$\lambda_1, \lambda_2, \lambda_3, \lambda_5$	0.789	0.604	0.527	0.789	0.497	0.477	0.847	0.643	0.609
$\lambda_1, \lambda_2, \lambda_4, \lambda_5$	0.778	0.576	0.495	0.789	0.500	0.475	0.847	0.643	0.609
$\lambda_1, \lambda_3, \lambda_4, \lambda_5$	0.778	0.576	0.495	0.789	0.497	0.477	0.847	0.645	0.615
$\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$	<b>0.800</b>	<b>0.616</b>	<b>0.535</b>	<b>0.811</b>	0.547	<b>0.523</b>	<b>0.863</b>	<b>0.695</b>	<b>0.625</b>

The values in bold are the highest ones

from Network Analysis theory that there are four main centrality measures, namely Degree Centrality, Closeness Centrality, Betweenness Centrality and Eigenvector Centrality [50]. However, only the former is fast to compute, so that it has a low impact on the training time of the GNN; instead, the others have a long computation time. Therefore, we chose Weighted Degree Centrality as the benchmark centrality measure. Regarding the choice of the main nodes, we know that Degree Centrality follows a power law distribution [50]. Therefore, we thought of using the Pareto principle underlying such a distribution and selected the 20% of nodes with the highest Weighted Degree Centrality values as the most important nodes.

At this point, similar to what we did for the previous measures, we trained FSGNN for 100 epochs. At the end of each epoch, we calculated  $\tau_{wd}$  (i.e., the Kendall's tau coefficient specialized to Weighted Degree Centrality) for each node in the training split and then averaged the values thus obtained. The corresponding results are reported in Fig. 4.

From the analysis of this figure, we observe that  $\tau_{wd}$  tends to decrease rapidly during the training of FSGNN. In fact, in all cases, we start with  $\tau_{wd} = 1$  and, in a few epochs, reach a value of  $\tau_{wd}$  between 0 (for Cora, Chameleon, Actor and Cornell) and 0.4 (for Texas and Wisconsin). This result shows that the rankings of the most central nodes of  $G$  and  $G^e(t)$  are different. This tells us that, during the training of FSGNN, the weighted degree centrality of the nodes of  $G^e(t)$ , and consequently the weights of the edges incident on them, substantially change.

After that, we wanted to test whether these changes had a positive impact on the training of FSGNN. To this end,

similar to what we have done for the previous measures, we employed the specialization  $\mathcal{L}_\gamma^e$  of the function  $\mathcal{L}^e$  defined in Eq. 3.10. Since, in this experiment, we chose the Weighted Degree Centrality as centrality measure, for the sake of clarity, we prefer to write  $\mathcal{L}_{wd}^e$  and  $\tau_{wd}$ , instead of  $\mathcal{L}_\gamma^e$  and  $\tau_\gamma$ .  $\mathcal{L}_{wd}^e$  is defined as:  $\mathcal{L}_{wd}^e = (1 - \lambda) \mathcal{L} - \lambda \tau_{wd}$ . We want to point out again that this is only a change of notation to increase the clarity of presentation. We repeated this experiment five times with different training and testing splits. The average results thus obtained are reported in Tables 9 and 10.

From the analysis of these tables, we can see that  $\tau_{wd}$  has a positive impact on the training of FSGNN on average. In particular, as for Cora, there is a small increase in Precision and Recall. In the case of Chameleon, improvement is negligible. As for Actor, Accuracy increased by 2.1% (for  $\lambda = 0.9$ ), Precision by 15.6% (for  $\lambda = 0.4$ ) and Recall by 2.3% (for  $\lambda = 0.9$ ). Instead, as for Cornell, Accuracy increased by 7.2%, Precision by 9.1% and Recall by 11.4% when  $\lambda = 0.8$ . In Texas, Accuracy grew by 10.2%, Precision by 15.3% and Recall by 21.1% when  $\lambda = 0.8$ . Finally, in Wisconsin, Accuracy increased by 3.7% (for  $\lambda = 0.8$ ), Precision by 7.4% (for  $\lambda = 0.9$ ) and Recall by 5.7% (for  $\lambda = 0.8$ ). As a consequence, we can conclude that, in our setting, this measure also has an impact on the training of FSGNN because in almost all cases its introduction into the loss function resulted in an improvement in classification performance compared with the baseline.

#### 4.6 Experiments on the structural information in the GNN training

Having tested the improvements in the training of a GNN made by each analysis measure separately from the others, we now want to verify the effectiveness of  $\mathcal{L}^e$  (see Eq. 3.5), which takes into account the contributions of all the analysis measures at once. Recall that, in the definition of  $\mathcal{L}^e$ , each of the weights  $\lambda_i$ ,  $1 \leq i \leq 5$ , associated with  $\mathcal{L}$  and the four analysis measures under consideration, ranges in the real interval  $[0, 1]$ , and  $\sum_{i=1}^5 \lambda_i = 1$ . Therefore, by adjusting these weights, we could define many versions of  $\mathcal{L}^e$  according to our needs (see, for instance, Eqs. 3.7–3.10). In all the previous tests, we considered scenarios in which only two values of  $\lambda_i$  were different from 0; one of them was always  $\lambda_1$  while the other depended on the analysis measure we wanted to test. It would now be interesting to consider 3, 4 and 5 weights  $\lambda_i$  different from 0 in  $\mathcal{L}^e$ . Unfortunately, testing all possible combinations of  $\lambda_i$  is extremely wasteful and essentially useless. Therefore, in conducting the experiments described in this section, we had to find a workaround different from the one of the previous experiments.

To this end, we decided to add the weights  $\lambda_i$  as learnable parameters of FSGNN. In this way, their values are learned during the training of the model. As a result, we do not have to check all the values but, hopefully, we obtain the best ones by solving the node classification task. Moreover, in all the previous experiments, we have seen that excluding the contribution of the initial loss function  $\mathcal{L}$  often returned very low results (see the last rows of Tables 3, 4, 5, 6, 7, 8, 9, 10). This highlights that  $\mathcal{L}$  is critical to the Machine Learning task performed by FSGNN. Consequently, we decided to keep  $\lambda_1$  (i.e., the weight associated with  $\mathcal{L}$  in  $\mathcal{L}^e$ ) in all tests. The results of the experiments for the various combinations of  $\lambda_i$  kept in the test and the corresponding performances obtained by FSGNN are shown in Tables 11 and 12.

In these tables, the row “Baseline” corresponds to the case in which we train FSGNN with only the loss function  $\mathcal{L}$  and, therefore, to the case in which  $\lambda_1 = 1$ ,  $\lambda_i = 0$ ,  $2 \leq i \leq 5$ . Each row corresponds to a choice of the parameters  $\lambda_i$  that we decide to keep and train. The values of the parameters not present in that row are set to 0. For example, in the second row of the two tables, we decided to keep only the weights  $\lambda_1$ ,  $\lambda_2$  and  $\lambda_3$ ; consequently,  $\lambda_4 = 0$  and  $\lambda_5 = 0$ . The values of the three weights  $\lambda_1$ ,  $\lambda_2$  and  $\lambda_3$  are initially set equal to each other and, consequently, equal to  $\frac{1}{3}$ . At each iteration of the training, the values of the weights vary. To give an example, let us consider the last row of Table 12. It represents a scenario in which all the weights  $\lambda_i$ ,  $1 \leq i \leq 5$ , are kept and their values are tuned

during training. At the first iteration of training,  $\lambda_i = 0.2$ ,  $1 \leq i \leq 5$ . Such a scenario returns the maximum values of Accuracy, Precision and Recall for the Cornell and Wisconsin datasets, the maximum value of Precision for the Actor dataset and the maximum values of Accuracy and Recall for the Texas dataset. Regarding the values of the weights  $\lambda_i$ ,  $1 \leq i \leq 5$ , in this row we obtained that: (i) for the Cornell dataset:  $\lambda_1 = 0.12$ ,  $\lambda_2 = 0.22$ ,  $\lambda_3 = 0.22$ ,  $\lambda_4 = 0.22$ ,  $\lambda_5 = 0.22$ ; (ii) for the Texas dataset:  $\lambda_1 = 0.12$ ,  $\lambda_2 = 0.25$ ,  $\lambda_3 = 0.25$ ,  $\lambda_4 = 0.23$ ,  $\lambda_5 = 0.15$ ; (iii) for the Wisconsin dataset:  $\lambda_1 = 0.12$ ,  $\lambda_2 = 0.23$ ,  $\lambda_3 = 0.22$ ,  $\lambda_4 = 0.22$ ,  $\lambda_5 = 0.20$ ; (iv) for the Actor dataset:  $\lambda_1 = 0.12$ ,  $\lambda_2 = 0.24$ ,  $\lambda_3 = 0.24$ ,  $\lambda_4 = 0.23$ ,  $\lambda_5 = 0.18$ .

From the analysis of Tables 11 and 12, we observe some interesting results. First of all, they are in line with the ones seen in the previous tests, although we employed a completely different workaround in this experiment. In particular, similar to the previous tests, the improvements achieved by our framework when applied to Cora and Chameleon are slight. As a matter of fact, the performances achieved are similar to that of the baseline, which means that the framework did not alter the training of FSGNN too much. As for Actor, the performances achieved are similar to that of the baseline with the exception of Precision, which, with the use of all five weights, achieves a substantial increase. We observe that FSGNN has the most significant improvements for Cornell, Texas, and Wisconsin. These are smaller datasets than the previous ones and all refer to the same context. These observations might prompt some preliminary considerations about the role that dataset size can play on the behavior of our framework. However, before coming to more definitive conclusions, we feel it appropriate to explore this aspect further in the future.

More specifically, the results obtained by our framework using Actor, Cornell, Texas and Wisconsin are promising. In fact, in almost all cases, the values of the classification metrics are greater than those of the baselines. In particular, as for Actor, while Accuracy remains the same, Precision increases by 13.77% using all five weights. Recall increases by 1.33% when we use the weights  $\lambda_1$ ,  $\lambda_2$ ,  $\lambda_3$  and  $\lambda_5$ ; if we use all five weights Recall still increase, but its growth is smaller, as it is 0.67%. As for Cornell, the highest values of Accuracy, Precision and Recall are obtained using all five weights, and thus all the analysis measures. In this case, compared to the baseline, the values of Accuracy, Precision and Recall increase by 5.68%, 10.00% and 7.86%, respectively. As for Texas, the highest values of Accuracy and Recall are obtained using all five weights. In this case, compared to the baseline, Accuracy increases by 6.99%, Precision by 5.60% and Recall by 17.00%. Actually, for this dataset, there are other configurations that are able to provide an even greater increase in Precision, equal

to 10.81%. Finally, as for Wisconsin, the highest values of the classification metrics are again obtained using all five weights. In this case, compared to the baseline, Accuracy increases by 3.23%, Precision by 9.45% and Recall by 5.22%.

In conclusion, this experiment shows that our framework is capable of ensuring substantially higher classification performance than the baseline for the three datasets with smaller sizes. Currently, we think it is possible to hypothesize a role of dataset size on the performance of our approach. However, we think it is premature to draw firm conclusions about this. Certainly, the results obtained can be a significant starting point for further investigations in this direction.

## 5 Conclusion

In this paper, we have proposed a framework that employs the theory and techniques of Network Analysis to investigate the dynamics underlying the learned representations of a GNN. Our framework receives a graph as input and passes it to the GNN to be analyzed. This returns the suitable node embeddings corresponding to the graph received as input. Afterward, our framework uses the original graph and the corresponding node embeddings to derive insights concerning the behavior of the GNN. Then, it employs these insights to define a new loss function that accounts for the differences between the graph received as input and the one reconstructed from the embeddings returned by the GNN. Finally, it uses this loss function to enhance the training of the GNN in such a way as to improve its performance. We have also described a large set of experiments that confirmed the goodness of our framework.

The main contributions of this paper with respect to the existing literature are as follows: (i) it proposes a method to map the learned representations returned by a GNN onto the graph from which they were obtained; (ii) it uses the theory and techniques of Network Analysis to define a framework that, in many datasets of different size and nature, has been shown capable of extracting insights regarding the structure and behavior of the GNN; (iii) it defines a new loss function that, in various datasets of different size and nature, assessed and enhanced the quality of the learning process of the GNN so that it can subsequently return better results.

This paper should not be considered as an ending point but rather as a starting point for further future research on this topic. In particular, in future, we would like to improve the representation of  $G^e(t)$  using a multilayer network [60] in order to be able to handle more details about the embeddings returned by the GNN during its training. The

multilayer network representation could have a temporal component corresponding to the training epochs. Each layer could be associated with  $G^e(t)$  at a given epoch  $t$ , and layers could be connected to each other according to the training progress. As a further development, we would like to improve our framework so that it can handle other GNN architectures, such as Graph Autoencoder and Spatio-Temporal Graph Neural Networks. These two types of models bring new challenges, and it would be interesting to see if, how, and with what modifications our framework could address them. Last but not least, we would like to test our framework on other machine learning tasks where GNNs are used, such as graph classification, edge prediction and unsupervised scenarios.

**Funding** Open access funding provided by Università Politecnica delle Marche within the CRUI-CARE Agreement.

**Data availability** The dataset used for our study is publicly available at the link <https://github.com/lucav48/gnn-sna/>.

## Declarations

**Conflict of interest** The authors declare that they have no conflict of interest.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

1. 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
2. Bronstein M, Bruna J, LeCun Y, Szlam A, Vandergheynst P (2017) Geometric deep learning: going beyond Euclidean data. *IEEE Signal Process Mag* 34(4):18–42
3. Zhou Y, Zheng H, Huang X, Hao S, Li D, Zhao J (2022) Graph neural networks: taxonomy, advances, and trends. *ACM Trans Intell Syst Technol* 13(1):1–54
4. Yu C, Deng G, Gui N (2023) PairGNNs: enabling graph neural networks with pair-based view. *Neural Comput Appl* 35(4):3343–3355
5. Kang S (2021) Product failure prediction with missing data using graph neural networks. *Neural Comput Appl* 33(12):7225–7234

6. Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, Wang L, Li C, Sun M (2020) Graph neural networks: a review of methods and applications. *AI Open* 1:57–81
7. Zhang B, Guo X, Tu Z, Zhang J (2022) Graph alternate learning for robust graph neural networks in node classification. *Neural Comput Appl* 34(11):8723–8735
8. Wang X, Jin B, Du Y, Cui P, Tan Y, Yang Y (2021) One-class graph neural networks for anomaly detection in attributed networks. *Neural Comput Appl* 33:12073–12085
9. Arazzi M, Cotogni M, Nocera A, Virgili L (2023) Predicting tweet engagement with graph neural networks. In: Proceedings of the ACM international conference on multimedia retrieval (ICMR'23), Thessaloniki, Greece, pp 172–180
10. Wu S, Sun F, Zhang W, Xie X, Cui B (2022) Graph neural networks in recommender systems: a survey. *ACM Comput Surv* 55(5):1–37
11. Zhang XM, Liang L, Liu L, Tang MJ (2021) Graph neural networks and their current applications in bioinformatics. *Front Genet* 12:690049
12. Reiser P, Neubert M, Eberhard A, Torresi L, Zhou C, Shao C, Metni H, van Hoesel C, Schopmans H, Sommer T, Friederich P (2022) Graph neural networks for materials science and chemistry. *Commun Mater* 3(1):93
13. Réau M, Renaud N, Xue LC, Bonvin AMJJ (2023) DeepRank-GNN: a graph neural network framework to learn patterns in protein-protein interfaces. *Bioinformatics* 39(1):1–8
14. Wein S, Malloni WM, Tomé AM, Frank SM, Henze G-I, Wüst S, Greenlee MW, Lang EW (2021) A graph neural network framework for causal inference in brain networks. *Sci Rep* 11(1):8061
15. Abadal S, Jain A, Guirado R, López-Alonso J, Alarcón E (2021) Computing graph neural networks: a survey from algorithms to accelerators. *ACM Comput Surv* 54(9):1–38
16. Liu Q, Nickel M, Kiela D (2019) Hyperbolic graph neural networks. In: Proceedings of the international conference on neural information processing systems (NeurIPS 2019), Neural Information Processing Systems Foundation, vol 32, Vancouver, Canada
17. Gu F, Chang H, Zhu W, Sojoudi S, El Ghaoui L (2020) Implicit graph neural networks. *Adv Neural Inf Process Syst* 33:11984–11995
18. Zhang C, Song D, Huang C, Swami A, Chawla NV (2019) Heterogeneous graph neural network. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD 2019) pages 793–803, Anchorage, AK, USA
19. Morris C, Ritzert M, Fey M, Hamilton WL, Lenssen JE, Rattan G, Grohe M (2019) Weisfeiler and Leman go neural: Higher-order graph neural networks. In: Proceedings of the AAAI conference on artificial intelligence (AAAI 2019) AAAI Press, Honolulu, HI, USA, 33, pp 4602–4609
20. Bodnar C, Frasca F, Wang Y, Otter N, Montufar GF, Lio P, Bronstein M (2021) Weisfeiler and Lehman go topological: Message passing simplicial networks. In: Proceedings of international conference on machine learning (ICML 2021) Virtual event, PMLR, pp 1026–1037
21. Xu K, Hu W, Leskovec J, Jegelka S (2019) How Powerful are graph neural networks? In: Proceedings of the international conference on learning representations (ICLR 2019), New Orleans, LA, USA. OpenReview.net. <https://openreview.net/forum?id=ryGs6iA5Kmq>
22. Wijesinghe A, Wang Q (2022) A new perspective on “How graph neural networks go beyond Weisfeiler-Lehman?”. In: Proceedings of the international conference on learning representations (ICLR 2022), Virtual event. OpenReview.net. [https://openreview.net/forum?id=uxgg9o7bI\\_3](https://openreview.net/forum?id=uxgg9o7bI_3)
23. Garg V, Jegelka S, Jaakkola T (2020) Generalization and representational limits of graph neural networks. In: Proceedings of the international conference on machine learning (ICML 2020), Virtual event, pp 3419–3430
24. Jegelka S (2022) Theory of graph neural networks: representation and learning. arXiv preprint [arXiv:2204.07697](https://arxiv.org/abs/2204.07697)
25. Yuan H, Yu H, Gui S, Ji S (2022) Explainability in graph neural networks: a taxonomic survey. *IEEE Trans Pattern Anal Mach Intell* 45(5):5782–5799
26. Balcilar M, Renton G, Héroux P, Gaüzère B, Adam S, Honeine P (2021) Analyzing the expressive power of graph neural networks in a spectral perspective. In: Proceedings of the international conference on learning representations (ICLR 2021), Virtual event. OpenReview.net. <https://openreview.net/forum?id=qh0M9XWxnv>
27. Lee S, Cheol Song B (2021) Interpretable Embedding Procedure Knowledge Transfer via Stacked Principal Component Analysis and Graph Neural Network. In: Proceedings of thirty-fifth AAAI conference on artificial intelligence (AAAI 2021) and thirty-third conference on innovative applications of artificial intelligence (IAAI 2021) and the eleventh symposium on educational advances in artificial intelligence (EAAI 2021), Virtual event. AAAI Press, pp 8297–8305
28. Heimann M, Safavi T, Koutra D (2019) Distribution of node embeddings as multiresolution features for graphs. In: Proceeding of the IEEE international conference on data mining (ICDM 2019), pp 289–298, Beijing, China. IEEE
29. Mara AC, Lijffijt J, Günemann S, De Bie T (2022) A systematic evaluation of node embedding robustness. In: Proceedings of learning on graphs conference (LoG 2022), Virtual event. PMLR, vol 198, pp 42
30. You J, Du T, Leskovec J (2022) ROLAND: graph learning framework for dynamic graphs. In: Proceedings of the ACM conference on knowledge discovery and data mining (SIGKDD 2022), pp 2358–2366, Washington DC, USA
31. Dehmamy N, Barabási A-L, Yu R (2019) Understanding the representation power of graph neural networks in learning graph topology. In: Proceedings of the international conference on neural information processing systems (NeurIPS 2019), Neural information processing systems foundation, pp 15387–15397, Vancouver, Canada
32. Lyu T, Zhang Y, Zhang Y (2017) Enhancing the network embedding quality with structural similarity. In: Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM 2017), pp 147–156, Singapore. ACM
33. Mendonça MRF, Barreto A, Ziviani A (2021) Approximating network centrality measures using node embedding and machine learning. *IEEE Trans Netw Sci Eng* 8(1):220–230
34. Wu J, He J, Xu J (2019) Demo-net: Degree-specific graph neural networks for node and graph classification. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD 2019), pp 406–415, Anchorage, AK, USA
35. Jin J, Heimann M, Jin D, Koutra D (2022) Toward understanding and evaluating structural node embeddings. *ACM Trans Knowl Discov Data* 16(3):58:1–58:32
36. Rizi FS, Granitzer M (2017) Properties of vector embeddings in social networks. *Algorithms* 10(4):109
37. Dalmia A, Ganesh J, Gupta M (2018) Towards Interpretation of Node Embeddings. In: Proceedings of the international workshop on learning representations for big networks (BigNet 2018), pp 945–952, Lyon, France
38. Dehghan-Kooshkghazi A, Kaminski B, Krainki L, Pralat P, Théberge F (2022) Evaluating node embeddings of complex networks. *J Compl Netw* 10(4):cnac030
39. Zhang Z, Xu Y, Cao Y, Yang L (2022) A graph convolution neural network-based framework for communication network

- k-terminal reliability estimation. *Secur Commun Netw* 1–14:2022
40. Munikoti S, Das L, Natarajan B (2022) Scalable graph neural network-based framework for identifying critical nodes and links in complex networks. *Neurocomputing* 468:211–221
  41. Min S, Gao Z, Peng J, Wang L, Qin K, Fang B (2021) STGSN: a spatial-temporal graph neural network framework for time-evolving social networks. *Knowl Based Syst* 214:106746
  42. Fan W, Ma Y, Li Q, Wang J, Cai G, Tang J, Yin D (2020) A graph neural network framework for social recommendations. *IEEE Trans Knowl Data Eng* 34(5):2033–2047
  43. Liu L, Wen G, Cao P, Hong T, Yang J, Zhang X, Zaiane OR (2023) BrainTGL: a dynamic graph representation learning model for brain network analysis. *Comput Biol Med* 153:106521
  44. Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. *J Stat Mech Theory Exp* 10:P10008
  45. Clauset A, Newman M, Moore C (2004) Finding community structure in very large networks. *Phys Rev Part E* 70(6):066111
  46. Raghavan UN, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. *Phys Rev E* 76(3):036106
  47. Harenberg S, Bello G, Gjeltema L, Ranshous S, Harlalka J, Seay R, Padmanabhan K, Samatova N (2014) Community detection in large-scale networks: a survey and empirical evaluation. *Wiley Interdiscip Rev Comput Stat* 6(6):426–439
  48. Newman MEJ (2006) Modularity and community structure in networks. *Proc Natl Acad Sci* 103(23):8577–8582
  49. Niwattanakul S, Singthongchai J, Naenudorn E, Wanapu S (2013) Using of Jaccard coefficient for keywords similarity. In: Proceedings of the international multiconference of engineers and computer scientists (IMECS'13), vol 1(6), pp 380–384
  50. Tsvetovat M, Kouznetsov A (2011) *Social network analysis for startups: finding connections on the social web*. Sebastopol, CA, USA. O'Reilly Media, Inc
  51. Saramäki J, Kivela M, Onnela J, Kaski K, Kertesz J (2007) Generalizations of the clustering coefficient to weighted complex networks. *Phys Rev E* 75(2):027105
  52. Das K, Samanta S, Pal M (2018) Study on centrality measures in social networks: a survey. *Soc Netw Anal Min* 8:1–11
  53. Samara B, Randles RH (1988) A test for correlation based on Kendall's tau. *Commun Stat Theory Methods* 17(9):3191–3205
  54. Wang Q, Ma Y, Zhao K, Tian Y (2020) A comprehensive survey of loss functions in machine learning. *Ann Data Sci*. <https://doi.org/10.1007/s40745-020-00253-5>
  55. Sen P, Namata G, Bilgic M, Getoor L, Galligher B, Eliassi-Rad T (2008) Collective classification in network data. *AI Mag* 29(3):93–93
  56. Rozemberczki B, Allen C, Sarkar R (2021) Multi-scale attributed node embedding. *J Compl Netw* 9(2):cnab014
  57. Tang J, Sun J, Wang C, Yang Z (2009) Social influence analysis in large-scale networks. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD'09), pp 807–816, Paris, France
  58. Pei H, Wei B, Chang KC, Lei Y, Yang B (2020) Geom-GCN: geometric graph convolutional networks. arXiv preprint [arXiv:2002.05287](https://arxiv.org/abs/2002.05287)
  59. Maurya SK, Liu X, Murata T (2021) Improving graph neural networks with simple architecture design. arXiv preprint [arXiv:2105.07634](https://arxiv.org/abs/2105.07634)
  60. Kivela M, Arenas A, Barthelemy M, Gleeson JP, Moreno Y, Porter MA (2014) Multilayer networks. *J Compl Netw* 2(3):203–271

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.