Knowledge-Based Systems 229 (2021) 107345

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A deep learning approach for semi-supervised community detection in Online Social Networks

Aniello De Santo[a], Antonio Galli[b], Vincenzo Moscato[b], Giancarlo Sperlì[b,*]

[a] Department of Linguistics of the University of Utah, Salt Lake City, UT 84112, USA
[b] Department of Electrical Engineering and Information Technology (DIETI), University of Naples "Federico II", Via Claudio 21, Naples, Italy

ARTICLE INFO

Article history:
Received 6 April 2021
Received in revised form 11 June 2021
Accepted 24 July 2021
Available online 28 July 2021

Keywords:
Social Network Analysis
Semi-supervised community detection
Online Social Networks
Deep learning

ABSTRACT

Social Network Analysis (SNA) has gained popularity as a way to unveil and identify useful social patterns as *communities* among users. However the continuous, exponential growth of these networks (both in terms of number of users, and in terms of the variety of different interactions that these networks allow) has made the development of efficient and effective community detection techniques a challenging computational task. In this paper, we propose an innovative approach for *Semi-supervised Community Detection*, exploiting Convolutional Neural Networks to simultaneously leverage different properties of a network — such as topological and context information. Crucially, computational cost is optimized by building on the insight that representing network connections over particular sparse matrices can significantly decrease the number of operations that need to be explicitly performed. By extensively evaluating our system on large (artificial and real-world) datasets, we show that our approach outperforms a variety of existing state-of-the-art techniques in terms of running time, as well as over $Macro-$ and $Micro-F1$.

1. Introduction

On-line Social Networks (OSNs) are nowadays a central hub for human interaction and information sharing, their popularity and variety having grown more and more in the last decade. The exponential increase in the amount of people interacting over these platforms has obviously lead to a change in the patterns of usage registered across OSNs, as well as the type and amount of information generated from them. For instance, Facebook reported 1,8 billions of active daily users producing hundreds of thousands of comments and posts every 60 seconds [1].

Because of the complex interaction between large amount of users and the heterogeneous information they produce, OSNs are of interest not only to different kinds of companies — attracted by the possibility of increasing profit through targeted, real-time and well-timed user-oriented actions [2,3] — but also to a variety of researchers inspired by the technological challenges offered by the study of users' social behavior within large online communities [4]. In particular, *Social Network Analysis* (SNA) can help develop methodologies to effectively explore the different social "ties" among users within such environments for a variety of applications: viral marketing, expert finding, community detection, influence analysis, social recommendation, and so on.

In this sense, modern OSNs seem to be characterized by the tendency of individuals to associate and bond with others who share similar interests (i.e. *homophily*), as in the proverb "birds of a feather flock together". In turn, this leads to the spontaneous formation of smaller user *communities* within larger OSNs with varying degrees of internal cohesiveness.

Accurate *community detection* is relevant to a variety of SNA applications [4–6]. For instance, discovering hubs and leaders ("influencers"), or experts for a given subject is an easier task when the search takes place over a well-structured community of users with similar interests. Uncovering networks of users clustering with respect to specific topics can also help maximize the spread of new technologies [7,8]. Similarly, recommendation systems can be made more reliable when considering how social ties influence users' choices and behavior within a community [9,10]. Recognizing outlier users inside a specific community can also significantly contribute to the detection of undesirable behavior (from spam bots to hate speech).

More generally, community detection techniques over large user groups can be generalized easily to any complex system that can be conceptualized as a network. In this sense, modern industrial systems are often modeled as complex networks, with flows of information affecting reliability and operational performance. Thus, companies organized around complex internal structures generating significant amount of data could leverage community detection techniques to improve productivity, balance costs and

* Corresponding author.
E-mail addresses: aniello.desanto@utah.edu (A. De Santo), antonio.galli@unina.it (A. Galli), vincenzo.moscato@unina.it (V. Moscato), giancarlo.sperli@unina.it (G. Sperlì).

https://doi.org/10.1016/j.knosys.2021.107345
0950-7051/© 2021 Elsevier B.V. All rights reserved.

benefits, predict performance degradation, and optimize service needs.

While effectively unveiling users' communities on the base of their interests and social connections seems then to be one of the most important goals (and challenges) for state-of-the-art SNA, it is hard to find in the literature a universally accepted definition of *community* [11–13]. At its core though, the term community surely has to deal with social context: people naturally tend to create groups within a social environment. Importantly, a required property of a community is *cohesiveness*: users of the same community are strongly connected to each other. The simplest approach is then define an OSN community as a group of users who share interests, by generating/sharing similar content or interacting with each other more frequently than with other users in the network.

A variety of techniques have been proposed to deal with the community detection problem — leveraging, among others, Game Theory [14], network topological features [15], and greedy methods [16]. Crucially, while the majority of existing approaches for community detection exploit network topology and structure, the essential role played by other background or context information in defining such communities has often been ignored.

Prior information about users' behavior and interactions can provide fundamental insights into the structure of real-word OSN communities. *Semi-supervised community detection* techniques focus on the best way to use prior information to support the discovery process of community structure. In these approaches, there are nodes for which the true community assignment is known in advance — the problem then is how to find the correct communities for unlabeled nodes within a social graph according to a label propagation process.

Early attempts to address this problem have relied on the use of matrix factorization techniques [17,18]. While these techniques seem to successfully capture core information about the network, the significant cost of working over high dimensional matrices makes them not suitable for real applications over modern OSNs. Because of this, the most popular approaches to semi-supervised community detection in the last couple of years have been relying on network embeddings [19,20] — in order to learn low-dimensional representations for nodes in a network — and convolutional neural networks (CNNs), extending the more general problem of semi-supervised classification to graph nodes of a social network [21].

In sum, given the ever-growing status of OSNs networks, one of the major challenges of community detection is how to efficiently keep track of both global information about the network of interactions, and local information about (sub-communities of) users.

With this in mind, this paper presents a semi-supervised community detection approach, combining deep learning techniques with topological properties of a social network. Specifically, our approach exploits *Convolutional Neural Networks*, in order to leverage prior information about the network. As mentioned, user interactions within different OSNs can be straightforwardly represented as sparse, high dimensional adjacency matrices. However, the dimensionality of modern OSNs is an issue for CNNs, which are traditionally used to classify images with significantly smaller *input matrices*. In order to deal with the dimensionality and sparsity issues, and efficiently perform a convolution on a very large sparse matrix, we opportunely modify the standard CNN's input layer by introducing the *SparseConv2D* operator. The efficacy and effectiveness of the approach is then evaluated using data from artificial simulated and real-world networks. To the best of our knowledge, we are the first to adapt current CNNs over very sparse matrices, thus fruitfully extending well-studied techniques to the community detection problem.

The rest of the paper is organized as follows. Section 2 reviews recent, popular approaches to community detection, with particular focus on deep learning based methods. Section 3 details this paper's approach to the semi-supervised community detection problem, and explores the issue of high dimensionality and sparsity of adjacency matrices. Section 4 presents the functional architecture of the system, with several implementation details. Finally, Sections 5 and 6 discuss the experimental results and the broader impact of our methodology.

2. Related work

As mentioned above, although community detection has become a core application of SNA, the term "community" still lacks a single, unique definition. For instance, while from a topological point of view a *community* can be seen as a subset of nodes densely connected with respect to others over a social graph [13,22–24], other work has pointed out the relevance of semantic features, defining a community as a subset of nodes sharing common properties and/or playing similar roles within the social graph [25]. Crucially, while these two perspectives are not incompatible, the task of merging the best of the two is far from trivial from a computational perspective — for instance, the full enumeration of community substructures within a network is an NP-Complete problem [26].

So far, community analysis [27] has been generally based on two fundamental steps: (i) detection of meaningful communities within a network; and (ii) evaluation of identified subgroups with measures that depend on the chosen concept of community.

There are several state-of-the-art methodologies addressing the identification problem from a topological perspective. For instance, *SCAN* [28] — which aims to identify sub-graphs satisfying specific conditions for what can be considered a community — or *FastGreedy* [29] which defines the vertices of a community as a graph through the optimization of particular geometric measures (i.e. normalized cut, conductance, modularity, etc.). Building on these ideas, other approaches combine both global and local topological information using label propagation strategies [30,31]. Another class of community detection algorithms relies on matrix factorization methods. Liu et al. [32] propose a semi-supervised approach that combines graph regularization with pairwise constraints on the basis of node popularity. Wu et al. [33] adopt instead a non-negative matrix factorization method, using hypergraph regularization.

Importantly, these approaches rely on knowledge about the entire network matrix, making them more suitable for small datasets — an issue partially addressed by leveraging statistical models to split a network into local communities [34].

A different way of exploiting topological features has been to look at community detection as a *Team Foundation* problem, combining knowledge about the structure of the network with semantic information about established connections between nodes. For instance, focusing on scientific collaboration networks, Mercorio et al. [35] propose an algorithm based on Node Location Analysis over a unified model combining scholarly document metadata with semantic information. Similarly, Najaflou et al. [36] show the value of domain specific knowledge by introducing Chemistry and Expertise Level metrics, that respectively measure the communication scale required by a task and the overall expertise among potential teams.

As in many other areas of information technology, the quick rise of deep learning methods in the last few years has also spread to community detection in OSNs. For example, Lin and Coen [37] propose a learning method for deep neural networks based on random graph walk, by using authoritative instances as training seeds to reduce the amount of required labeled data. In [38], the authors analyze *inter-community interactions* in order to evaluate conflicts among communities, mitigating the negative impact of conflictual interactions and predicting possible conflicts by using a Long short-term memory (LSTM) model that combines user, community, and text features.

Semi-supervised approaches have been shown to be a good way to integrate network topology with prior information for community detection [39], and deal with dimensionality issues. In this sense, Yang et al. [40] propose a method based on a modularity function and a low-dimensional embedding matrix computed by using an auto-encoder schema. Bruna et al. [41] propose a *Graph Neural Network* (GNN) model with the non-Backtracking operator defined on the line graph of edge adjacency. Alternatively, some work has instead focused on *graph embeddings* in order to map a graph into a low-level dimensional space (see [42] for details). Wang et al. [43] present a semi-supervised deep model — *Structural Deep Network Embedding* (*SDNE*) — to preserve both local and global information about the structure of a network by jointly optimizing first and second order proximity while at the same time making the method robust with respect to sparse networks.

Moving away from supervised methods, Perozzi et al. [19]'s *DeepWalk* allow for latent representations of social graph relations using a *random walk generator*, coupled with *SkipGram* for sampling random vertices and updating the representation, respectively. Extending DeepWalk with a controlled path sampling process, *Node2Vec* [20] improves the learning of latent representations with a likelihood maximization function aimed at preserving network neighborhoods of nodes. Relatedly, Tang et al. [44] propose *LINE*, a network embedding method, leveraging edge-sampling to deal with stochastic gradient descent limitations, and preserve both local and global graph structure. Among other network embedding approaches, Wang et al. [45] use metapath based neighbors encoding hierarchical information, while Zhu et al. [21] perform random walks over network graphs to build a coarsened graph with information about different edge types. In this latter approach, the coarsened graph's features are then used in conjunction with the original graph as the input to a H-GNN module. Cavallari et al. [46] propose *ComE*, a framework based on a particular setting of graph embeddings targeting embedding communities instead of individual nodes. In particular, each community can be modeled as a multivariate Gaussian distribution in the 2D space and the learning of the graph structure hinges upon a closed loop among community embeddings, community detection, and node embeddings. Similarly, in order to learn node representations, it is possible to feed the positive pointwise mutual information matrix directly to deep stacked sparse autoencoders [47].

Finally, adversarial approaches have also been gaining popularity. For instance, Wang et al. [48] propose *GraphGAN*, a graph representation learning framework leveraging generative adversarial networks. Along these lines, Wang et al. [49] introduced *CANE* in order to simultaneously learn node representations and identify network communities.

Table 1 summarizes the approaches overviewed here, with a brief description of advantages and limits of each. In particular, the "Dataset & Performances" column lists the datasets each approach was tested over, reporting performances in terms of the metrics detailed in the "Metrics" column. The discussion so far highlighted how, while the variety of methodologies that have been applied to the issue of community detection is inspiring and encouraging, there is still room for vast improvement from several directions. In particular, the majority of the approaches overviewed here rely on a pre-processing stage on the OSNs under investigation, so to extract relevant topological/semantic features, or to generate latent representations that can subsequently be used as input to a complex neural network.

Taking into account the most recent results in the literature, in this paper we propose a semi-supervised approach for community detection in OSNs based on deep learning techniques. However, departing from previous approaches, we avoid preprocessing and consider the whole network as the real input to the system. This allows us to fully preserve both local and global structural information about a network's graph, fully exploiting the topological characteristics of the graph itself, and smartly addressing the scalability issues for the high dimensionality of the adjacency matrix. Specifically, using the entire adjacency matrix, we preserve all proximity relationships — in contrast to other approaches guaranteeing second-order proximity at best [20,43,44].

3. Convolutional neural networks over sparse matrices

This paper approaches the problem of community detection within OSNs by expanding on previous semi-supervised deep learning techniques, and exploits *CNNs* with sparse input matrices.

CNNs are a well-known type of deep learning architecture, employing a mathematical operation called "convolution" to perform various tasks such as classification and segmentation. At its core, a simple CNN is a sequence of the following four layers: an input layer, a convolutional layer, a max-pooling layer, and a Fully Connected layer [50].

In our methodology, the CNN input layer handles a *user-to-user* matrix, whose elements can assume several meanings — such as the similarity measure of interest of two users with respect to the same contents, presence of a friendship relationship, and so on. More specifically, the network receives in input a two-dimensional adjacency matrix ($n \times n$, where n is the number of users). Slices of the adjacency matrix are obtained by extracting individual rows. Each row, represented as a vector, is then transformed into a matrix itself. The result is a set of n adjacency matrices, each matrix representing adjacency relationships between a specific user and the rest of the network.

3.1. Dimensionality issues

The core of our proposal is to exploit CNNs to assign users to appropriate communities, by leveraging prior network information (e.g. adjacency relations) in the training phase. Thus, we fundamentally rely on an adjacency matrix which explicitly represents the relationships among *all* users in an OSN. As discussed before, the high number of users in modern OSNs is a major challenge for computationally effective approaches. Obviously, the size of the adjacency matrix increases as the size of the network increases. However, users will have established relationships only with a relatively small subset of the actual network, thus resulting in high dimensional matrices that are fundamentally sparse.

Building on this insight, the idea at the core of this paper is to face the dimensionality issue by complementing the CNN approach with sparse matrix algebra. In particular, we propose a convolutional layer optimized for the computation of the convolution between high dimensional sparse matrices. The new layer performs the least possible number of convolutions, exclusively computing operations where the adjacency sub-matrix has effectively non-zero values. In addition, the pooling layer has been modified to deal with reduced feature maps from the convolution output.

Example 3.1. Fig. 1 shows a step by step decomposition of the computation of sparse convolutions. In this example, the adjacency matrix has only two non-zero elements, and thus convolutions are only performed over these values. Setting null values automatically – without explicit computations – for each other sub-component of the matrix.

Table 1
Summary of the state-of-the-art approaches reviewed in Section 2. The "Dataset & Performances" column contains the datasets each approach was evaluated over. Performance over each dataset is reported in terms of the metrics detailed in the "Metrics" column, as *Dataset(Score)*.

Reference	Pros	Cons	Metrics	Dataset & Performances
[37] (2010)	Learning method based on random walk using authoritative instance to reduce the amount of training data.		Macro average F1	UMBCBlog (0.91), AGBlog(0.93) Cora (0.78), MSPBlog (0.89) CiteSeer (0.68)
[38] (2018)	LSTM model combining user, community and text features to mitigate possible conflicts among communities.		Area Under Curve	Reddit (0.76)
[40] (2016)	Semi-supervised approach based on a modularity function and low-dimensional embeddings.		Normalized Mutual Information	Karate (1.00), Dolphins (0.889) Friendship6 (0.889), Friendship7 (0.888) Football (0.907), Pollbooks (0.927) Polbooks (0.552) Polblogs (0.389) Cora (0.463)
[39] (2015)	Semi-supervised learning framework combining network topology and prior information.	Perform a pre-processing stage for extracting relevant features or creating latent representation that are computed by a Neural Network failing to preserve proximity order.	Normalized Mutual Information	Karate (1.00), Dolphins (1.000) Friendship6 (0.965), Friendship7 (0.973) Football (0.942), Pollbooks (0.937)Polblogs (1.000)
[41] (2017)	Graph Neural Network model with the non-backtracking operator defined on the line graph.		Accuracy	Amazon (0.74), DBLP (0.78) Youtube (0.9)
[19] (2014)	Unsupervised learning method for representing social graph relations using a random walk generator and SkipGram.		Macro-F1 Micro-F1	BlogCatalog (0.289, 0.420), Flickr (0.246, 0.385) Youtube (0.354, 0.427)
[46] (2017)	Graph embedding in which the learning process hinges upon a closed loop among community embeddings, community detection and node embeddings.		Macro-F1 Micro-F1	BlogCatalog (0.324, 0.441), Flickr (0.268, 0.416) Wikipedia (0.112, 0.500), DBLP (0.924, 0.928) Karate Club (1.000, 1.000)
[45] (2019)	Node embedding approach using meta-path based Neighbors hierarchically.		Macro-F1 Micro-F1	DBLP (0.930, 0.939), IMDB (0.939, 0.543) ACM (0.906, 0.905)
[21] (2019)	Embedding model based on latent associations of different types of edges.		Accuracy	AIFB (0.972), MUTAG (0.823) BGS (0.931), AM (0.904)
[20] (2016)	Learning latent representation using random walk generator and SkipGram with a controlled path sampling process preserving a first order proximity.		Macro-F1 Micro-F1	BlogCatalog (0.290, 0.391), PPI (0.191, 0.242) Wikipedia (0.274, 0.581)
[44] (2015)	Network embedding based on a particular objective function for preserving first and second proximity order.		Macro-F1 Micro-F1	Wikipedia (0.836, 0.837), Flickr (0.257, 0.406) Youtube (0.362, 0.430), DBLP (0.651, 0.660)
[43] (2016)	Semi-supervised deep model for capturing the network structure preserving first and second proximity order.	Pre-processing is performed to extract relevant features preserving at maximum second order proximity.	Macro-F1 Micro-F1	BlogCatalog (0.312, 0.448), Flickr (0.261, 0.411) Youtube (0.373, 0.442), Arxiv (0.489, 0.576) 20newsgroup (0.566, 0.701)
[48] (2018)	Learning an innovative graph representation leveraging generative adversarial networks and a novel *graph softmax*.		Accuracy Macro-F1	BlogCatalog (0.232, 0.330), arXiv AstroPh (0.855, 0.859) arXiv GrQc (0.849, 0.853), Wikipedia (0.213, 0.194) MovieLens 1M (0.298, 0.243)
[47] (2019)	Deep stacked sparse autoencoders able to learn the node representation.		Macro-F1 Micro-F1	BlogCatalog (0.328, 0.441), Flickr (27.01, 42.16) Cora (0.78220.7939)
[49] (2021)	*CANE* framework able to learn the node representation and identify the network communities using adversarial learning.		Accuracy Macro-F1	BlogCatalog (0.882, 0.334), arXiv AstroPh (0.885, 0.881) arXiv GrQc (0.861, 0.871), Wikipedia (0.879, 0.871)MovieLens 100k (0.921, 0.916) AmericanAir Traffic (0.917, 0.914), Twitter (0.881.0.877)
This paper (2021)	A semi-supervised approach combines topological and context information modeling OSN connections as sparse matrices to reduce computational costs.	The parameter tuning at its best can be computationally expensive due to the huge size of the input matrix.	Macro-F1 Micro-F1	BlogCatalog3 (0.351, 0.479), Flickr (0.297, 0.445), Youtube (0.419, 0.479)

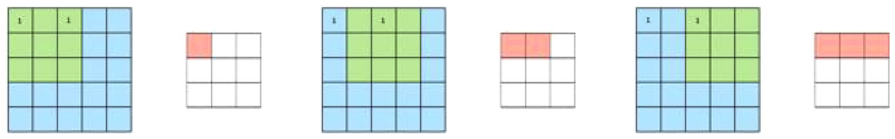


Fig. 1. An example of Sparse convolution. Given the sparsity of the matrix, convolution operations are performed exclusively around the non-zero elements (3 in total).

3.2. SparseConv2D

The *SparseConv2D* algorithm (Algorithm 1) illustrates how sparse convolutions are performed.

Rows 2–3 describe how to compute the dimension of the resulting feature maps, while row 4 illustrates a 180° rotation of the kernels. The loop of rows 5–7 takes each line of the sparse input matrix, each representing a single, individual node to be processed, and it performs a reshape operation in order to generate $w \times h$ matrices (where w and h are respectively reshaped weight and height) that are then added to the *SparseMatrix* list through the *Append* function. The matrices obtained in this way are still scattered and match with the adjacency matrices of individual users. At rows 8–11 the previously obtained adjacency matrices are iteratively selected. Once the mth master is taken, all the indexes corresponding to the positions of the non-zero elements are memorized through the *IndicesNonZero* function. Then, an empty list is initialized so to memorize the indexes that have carried out the convolution, thus avoiding performing the operation over the same values multiple times. The kernels received in input are iteratively selected and the convolution of each node is performed for each of the owned kernels (row 12). After selecting the mth adjacency matrix and the kth kernel, the positions to be convoluted must be identified (rows 13–14). In particular, the first loop provides the position of the next non-zero element on the adjacency matrix, while the second one provides all the indexes necessary to center the filter so that it matches the identified sub-matrix, as well as the position of the result within the feature maps. Rows 15–17 verify that the identified index has not already been used within the convolution.

Algorithm 1 SparseConv2D

```
Require: Sparse Matrix $m$, Matrix size $mSize$, Kernels $k$, Number of kernels $numk$, Reshaped width $w$, Reshaped height $h$
Ensure: $mSize$ matrices of dimension $(w - KernelSize + 1) \times (h - KernelSize + 1)$ representative of the Feature Maps

1: procedure SPARSECONV2D($m, mSize, k, numK, w, h$)
2:     $FMSizew \leftarrow w - size(k) + 1$
3:     $FMSizeh \leftarrow h - size(k) + 1$
4:     $Kernels \leftarrow Rotate180(k)$
5:     for $i \leftarrow 0$ to $mSize$ do
6:         $s \leftarrow$ Take the $i$-th row of $m$ and reshapes it into a sparse matrix $s$ of $w \times h$ dimension.
7:         $SparseMatrix \leftarrow Append(s)$
8:     for $m \leftarrow 0$ to $mSize$ do
9:         $rowMatrix \leftarrow SparseMatrix[m]$
10:        $NonZeroElement \leftarrow IndicesNonZero(rowMatrix)$
11:        $featureMapsIndices \leftarrow EMPTY$
12:        for $k \leftarrow 0$ to $numK$ do
13:            for each elements into $NonZeroElement$ do
14:                for each central position $(r_{index}, c_{index})$ do
15:                    if $r_{index}$ and $c_{index}$ not in $featureMapsIndices$ then
16:                        Selects from $m$-th $rowMatrix$ the sub-matrix get around the position element $(r_index, c_index)$. The dimension of sub-matrix is the same as kernel size.
17:                        Performs the convolution between sub-matrix and $k$-th kernel.
18:                        Appends $r_{index}$ and $c_{index}$ into $featureMapsIndices$.
19:                        Adds the convolution result obtained from node $m$ and $k - th$ kernel to the position $(r_{index}, c_{index})$ of the associated $featureMaps$.
20:    return $featureMaps$
```

If this is verified, the sub-matrix obtained around the non-zero index is selected and the convolution between that sub-matrix and the selected kernel is performed. The indexes of the element on which the convolution is realized are then added to a special list, keeping track of the elements the convolution has already been carried over. Finally, the result of the convolution is stored in the feature maps associated with the m-node and kth kernel, in the previously evaluated position (row 18–19). The return value will be the set of all the computed features maps (row 20).

It is worth noting that the initial weight W_{ij} at each layer can be computed according to the method shown in [51], assuming the biases are set to 0, with the following commonly used heuristic:

$$W_{ij} \sim U\left[\frac{-1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] \tag{1}$$

U being an uniform distribution in the defined interval, and n the number of W kernel matrix columns.

3.3. Max-pooling

A Max-pooling layer is generally applied on the feature maps generated from the convolutional layer to compute the maximum value for each patch in a map, creating a new set of the same number of pooled feature maps — each strictly of a smaller size of the original map. These reduced feature maps are then used within a second convolutional layer. Algorithm Algorithm 2 shows how to implement max-pooling for sparse feature maps.

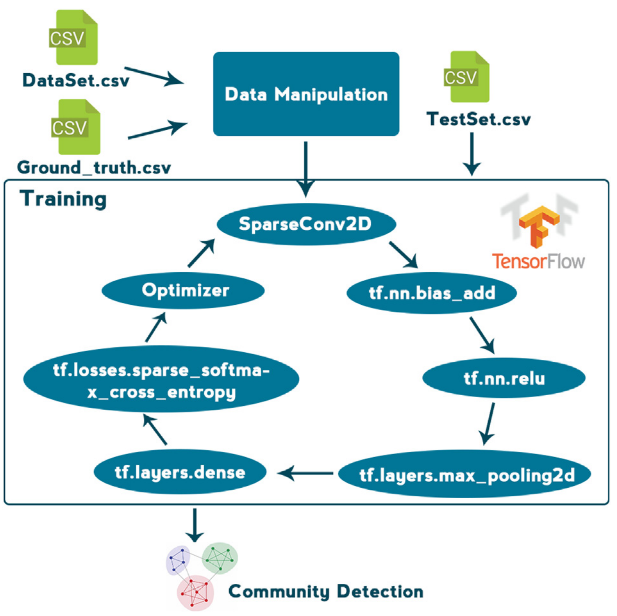


Fig. 2. CNN workflow with TensorFlow.

As shown at row 2, we first have to compute the number of non-zero values. Then, two loops are used to discover those non-zero values over which to perform max-pooling operations (rows 3–6). We identify the indexes of non-zero values, then we compute the max-pooling windows containing a particular value. We check if a max-pooling window has already been used in a previous step (row 7). We verify if non-zero values of a feature map are near to the matrix edges, in which case the max-pooling window will have to be smaller (rows 8, 10, 12 and 14). For each condition, there is a different feature map slice, according to the selected max-pooling window (9, 11, 13, and 15). If a non-zero value is far from the edges, the max-pooling window is $m1 \times m2$, where $m1$ and $m2$ are respectively its height and the weight. Rows 16–17 perform the actual max-pooling operations. First, the sparse matrix returned by the slicing function is "made" dense. Successively, we obtain the max element among $m1 \cdot m2$ elements. Finally, we compute the relevant indexes and extract the non-zero values (rows 18–19).

3.4. Complexity

The complexity of the convolution operation is tied to the size of the input and kernel matrices. In particular, let $M \times N$ and $k \times j$ be an input and a kernel matrix, respectively. The computational cost of performing the convolution operation is then fixed to $O(MNkj)$. As discussed in Section 3, our approach leverages the sparsity of input matrices and considers only non-zero elements. Thus, $O(MNkj)$ is the *worst* case for our modified algorithms, improving overall efficiency.

4. Architecture

Fig. 2 shows the workflow for our semi-supervised community detection system, together with the core details of the CNN structure (implemented using TensorFlow [52]).[1]

Data are essentially processed within the CNN, which performs n training step, minimizing the calculated loss. In particular, the *optimizer* (named *GradientDescentOptimizer*) is necessary

[1] https://www.tensorflow.org/.

Algorithm 2 MaxPooling

```
procedure MaxPooling(FeatureMap, m1, m2, f1, f2)
    NonZeroElements ← countNonZero(FeatureMap)
    for index ← 0 to NonZeroElements do
        indX, indY ← Indices(index, FeatureMap)
        XReg ← Int(indX/m1) * m1
        YReg ← Int(indY/m2) * m2
        if [XReg/m1,YReg/m2] not in indicesReducedFM then
            if XReg + m1 > f1 and YReg + m2 > f2 then
                reg ← Slice(FeatureMap, XReg, YReg, f1-XReg, f2-YReg)
            else if XReg + m1 > f1 then
                reg ← Slice(FeatureMap, XReg, YReg, f1-XReg, m2)
            else if YReg + m2 > f2 then
                reg ← Slice(FeatureMap, XReg, YReg, m1, f2-YReg)
            else
                reg ← Slice(FeatureMap, XReg, YReg, m1, m2)
            regionDense ← Dense(reg)
            maxElement ← MAX(regionDense)
            indicesReducedFM ← XReg/m1,YReg/m2
            valuesReducedFM ← maxElement
    return ReducedFeatureMap
```

to update a network's weights in order to reduce the loss function (*sparse softmax cross entropy*) by performing a back propagation on the SparseConv2D function. The training phase is depicted as a cycle in Fig. 2, since this is an iterative process aimed at identifying the best parameters of our model.

We then have two main components:

- Data Manipulation: Two CSVs are received as input, one related to the dataset, consisting of the global adjacency matrix for the network, and the other one corresponding to the ground truth. The adjacency matrix is saved as SparseTensor. We apply sparse_reshape over each row to obtain the adjacency matrix corresponding to each user.
- SparseConv2D: Corresponds to the algorithm described in 3.2, implemented through TensorFlow in Python. It performs the convolution operation with sparse matrices, replacing the existing Conv2D module of TensorFlow, which works for dense matrices.

This process results in a CNN trained for community detection with respect to the incoming dataset (training set).

5. Experimental evaluation

This section describes the set of experiments conducted in order to evaluate the efficiency, efficacy, and robustness of the proposed approach. For completeness, we compare the results of our methodology with respect to a variety of different baseline algorithms and datasets.

5.1. Experimental protocol

In order to evaluate the proposed approach, three types of experiments have been performed:

- Efficiency analysis: The efficiency of the SparseConv2D operation is evaluated in terms of running times with respect to matrices of different size and density degrees.
- Efficacy analysis: The efficacy of the proposed approach is compared to 8 baseline methods (DeepWalk [19], SDNE [43], LINE [44], SpectralClustering [53], Modularity [54], EdgeCluster [55], wvRN [56], and Majority).
- Model Robustness: We evaluate how the efficacy of the approach is affected by the number of deleted social graph edges.

We used five different datasets for our experiments: four existing datasets summarized in Table 2, and an artificial dataset generated in order to evaluate the performance of our system when varying parameters like density and matrix size.

The network model is composed of different layers: a SparseConv2D, a *bias_add*, a relu activation function, a Maxpooling2D, and a fully connected layer. In particular, the *SparseConv2D* algorithm is composed of different filters, whose size ranges from 3 to 5. The number of filters varies according to the network topology and size, while we use the *GradientDescentOptimizer* optimizer in order to minimize loss functions to train the network's weights. Furthermore, the optimization of the model's parameters is investigated in Sections 5.2 and 5.3 in terms of number of kernels, learning rate, optimizer, and size of feature maps.

Experiments were run on Google Colab equipped with one single core hyper threaded Xeon Processors @2.3Ghz, 12 GB of RAM and a Tesla K80 having 2496 CUDA cores and 12 GB GDDR5 VRAM. As mentioned, our system was implemented in Python 3.6 with TensorFlow 2.0 as back-end.

5.2. Efficiency analysis

This section details the efficiency of the SparseConv2D algorithm in terms of running time. In particular, experiments were performed on artificial datasets with the following characteristics:

- fixed density and variable size;
- variable density and fixed size.

This type of analysis was conducted over artificially generated datasets to better evaluate performance at different sizes and densities. The first set of experiment varied the size (number of nodes N) of the adjacency matrix from 10.000 to 200.000, with a density value ρ fixed to 10^{-4}. Results are shown in Fig. 3.

We compare our approach with respect to a *Divide et Impera* (DeI) strategy, which differs from ours in its use of the Conv2D operation instead of SparseConv2D to realize the CNNs. This comparison thus allowed us to explicitly evaluate the differences between SparseConv2D and Conv2D operations.

Table 2
Datasets' characterization ($|V|$, $|E|$ and $|Y|$ are respectively the number of vertices, edges and labels while ρ, CC, d and D(v) are the density, average cluster coefficient, diameter and average degree).

Name	$\|V\|$	$\|E\|$	$\|Y\|$	ρ	CC	d	D(v)
email-Eu-core	1,005	25,571	42	2.5×10^{-3}	0.3994	7	25.44
BlogCatalog3	10,312	333,983	39	6.3×10^{-3}	0.460	5	64.9
Flickr	80,513	5,899,882	195	1.8×10^{-3}	0.1652	3	146.7
YouTube	1,138,499	2,990,443	47	4.5×10^{-6}	0.0808	20	5.25

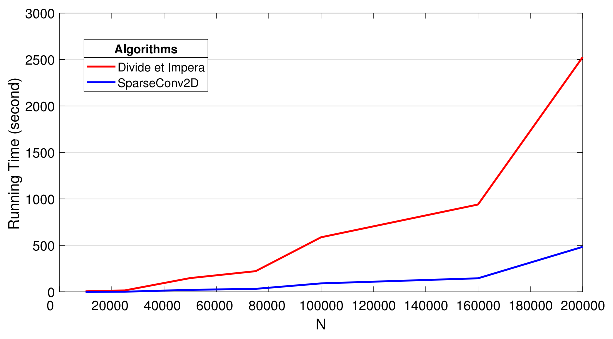


Fig. 3. This figure shows the running time variation of the SparseConv2D and DeI approach by varying the size of the network (number of nodes N) and fixed density ($\rho = 10^{-4}$).

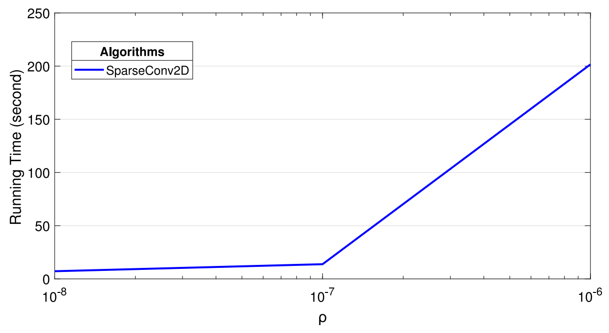


Fig. 5. SparseConv2D running times varying ρ ($N = 1000\,000$).

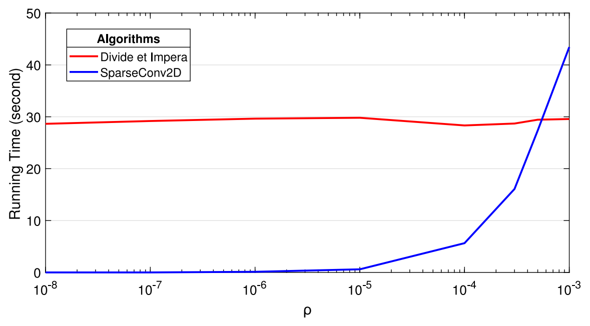


Fig. 4. Running time variation of SparseConv2D and DeI when varying the density of the network (ρ) and with fixed network size ($N = 50{,}000$).

As shown in Fig. 3, the approach based on SparseConv2D is faster when the matrix has a reduced size, in line with the fact that as the number of non-zero values increases, so does the number of convolutions to be performed. We evaluated ten runs in order to provide an estimate of the execution time with the relative variance. The average variance obtained with regards to all the executions is 9.72%.

In a second set of experiments, we varied the value for ρ from 10^{-8} to 10^{-3}, while the number of nodes was fixed to 50.000. Results are shown in Fig. 4.

Once again, the execution time is directly proportional to the number of non-zero values present in the adjacency matrix. It is easy to note that the SparseConv2D operation is convenient when the matrix is very sparse, while it is extremely slow with smaller degrees of sparsity. In contrast, the DeI approach has a constant behavior, as the number of elements it works on is always the same regardless of density.

Finally, to better highlight how the execution time of the SparseConv2D approach strongly depends on the number of non-zero values present in the matrix, we considered an adjacency matrix with a higher number of nodes ($1,000,000$), and different density values (varying from 10^{-8} to 10^{-6}). Results for this specific case are shown in Fig. 5.

This last case illustrates how even with matrices of extreme size, there are low density cases in which performance is particularly good — even outrunning performance over smaller datasets with higher density. Consider the case of 50,000 nodes and density equal to 10^{-4}, with a corresponding execution time of about 8 seconds, and contrast it with the corresponding case of $1,000,000$ nodes and density equal to 10^{-7}. In this latter case, running time is around the 3.62 s. Crucially, in the first case the number of non-zero values is equal to 250,000, while in the second case is 100,000. Thus, as expected, execution time is sensitive to the number of non-zero values, and not absolute size of the input matrix.

5.3. Efficacy analysis

The following experiments were performed using our CNN implementation, as described in Section 4, over three real world datasets: BlogCatalog3, Flickr and Youtube. We observe the *loss value*, measured at each step of network training, according to the following parameters characterizing the CNN:

- number of kernels;
- learning rate;
- type of optimizer.

The CNN model has been trained using the Tensorflow library's loss function *tf.losses.sparse_softmax_cross_entropy*. For each evaluation, training is performed by minimizing the loss value. Specifically, training ends when the difference between the loss values obtained in two successive steps is lower than a threshold value ϵ:

$$|loss_i - loss_{i+1}| < \epsilon$$

Furthermore, for the same optimizer and the same number of kernels, two different curves have been calculated, each with a different learning rate. To analyze the trend of the loss value when varying learning rate, the minimum number of steps that saturated the loss value of at least one of the two curves was considered as the total number of steps.

5.3.1. Gradient descent optimizer

We then evaluated the performance of the *gradient descent optimizer* through TensorFlow's basic functions *tf.train.GradientDescentOptimizer*. The gradient descent method is an iterative

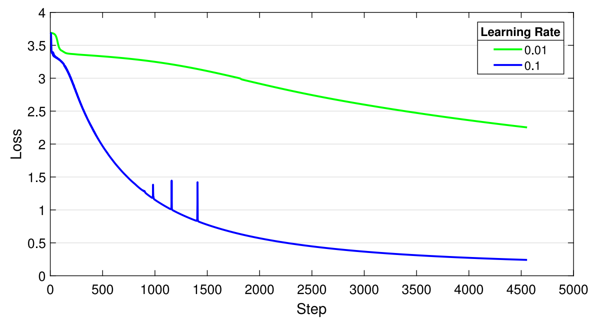


Fig. 6. CNN Gradient descent optimizer with 3 Kernels 3×3 .

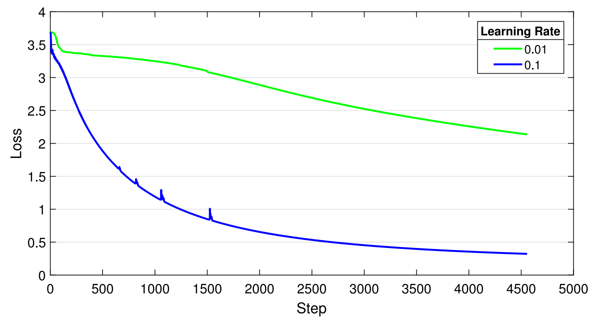


Fig. 7. CNN Gradient descent optimizer with 6 Kernels 3×3 .

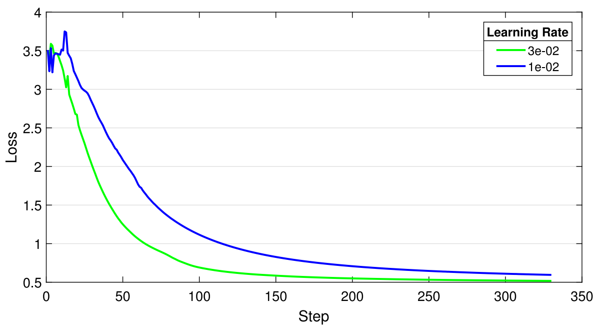


Fig. 8. Adam optimizer with 3 Kernels 3×3 .

method, which searches for the minimum/maximum of a function. In the case we considered, the goal is to minimize the loss function. The learning rates chosen for the experimentation are 0.1 and 0.01.

Fig. 6 shows the loss value trend when the number of kernels is 3, while Fig. 7 shows the same trend when the number of kernels is 6. From these Figures, it is easy to see how the two set ups exhibit similar trends, and thus that varying the number of kernels does not results in any significant changes.

Importantly, the curve corresponding to the gradient descent with learning rate 0.1 shows how the loss value is minimized faster than in the case of learning rate 0.01: after 4.000 *steps*, the loss value is approximately 0.3 in first case, while in the second one is 2.4.

5.3.2. Adam optimizer

The *Adam optimizer* is different from the gradient descent method. While the latter always keeps the learning rate constant, Adam adapts it.

Figs. 8 and 9 show the loss trend when the number of kernels is kept equal to 3 and 6, respectively. While the trend for the two

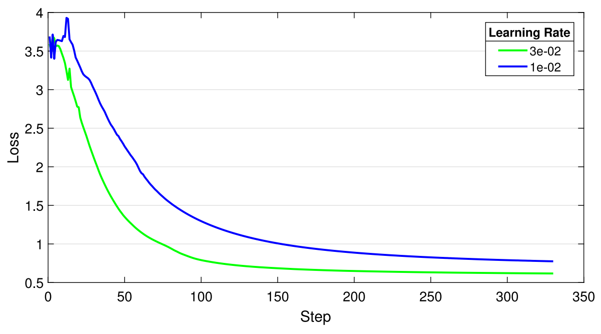


Fig. 9. Adam optimizer with 6 Kernels 3×3 .

configurations is overall very similar, with the first one we obtain a slightly lower loss given the same step.

As expected, the saturation of the loss occurs in fewer steps than with gradient descent. In fact, we reach a loss of about 0.6 after 150 steps using a learning rate of 0.03, while the number of steps to obtain the same loss value is about 300 considering a learning rate of 0.01. Generally, the loss value with the Adam optimizer does not fall below 0.5.

5.3.3. Evaluation metrics

Since the problem at hand is basically a multi-label classification task, we evaluate the performance of our system in terms of $Macro - F_1$ and $Micro - F_1$ [43,54].

Consider X as one of the possible labels. We refer to $TP(X)$, $FP(X)$, and $FN(X)$ as the number of true positives, false positives and false negatives, respectively. Assume then that C is the set of labels. $MacroF_1$ and $MicroF_1$ are defined as follows:

- $Micro - F_1$ assigns equal weight to each instance, defined as:

$$Pr = \frac{\sum_{X \in C} TP(X)}{\sum_{X \in C}(TP(X) + FP(X))} \tag{2}$$

$$R = \frac{\sum_{X \in C} TP(X)}{\sum_{X \in C}(TP(X) + FN(X))} \tag{3}$$

$$Micro - F1 = \frac{2 * Pr * R}{Pr + R} \tag{4}$$

where Pr is Precision and R Recall.
- $Macro - F_1$ assigns equal weight to each class, defined as:

$$Macro - F1 = \frac{\sum_{X \in C} F1(X)}{|C|} \tag{5}$$

where $F1(X)$ is the F1-measure for the label X.

5.3.4. Classification results

With all preliminaries in place, we can finally discuss the performance of our system in terms of effectiveness over different real-word datasets.

In particular, we built the training set via stratification on the adjacency matrix, so that the percentage of each class is balanced across both training and test set. As mentioned, performance was evaluated in terms of $Macro - F_1$ and $Micro - F_1$ for different training set sizes: 10%, 30%, 60% , 90% for the BlogCatalog3 dataset and 1%, 3%, 6% , 9% for the Flickr and Youtube datasets.

To evaluate the efficiency of the sparse CNN approach, we chose a variety of existing methods as baseline algorithms.

- **DeepWalk** [19]: An approach to learn latent representation of vertices in a network.

Table 3
Performance on BlogCatalog3.

	% Labeled Nodes	10%	30%	60%	90%
Micro-F1 (%)	CNN	30.51	39.67	44.70	47.92
	CANE	28.99	39.21	42.98	45.01
	GraphCAN	28.78	39.01	42.71	44.76
	ComE	27.18	38.64	41.68	44.12
	DNNNC	28.15	38.91	41.89	44.59
	SDNE	31.11	36.70	41.88	44.88
	LINE	30.43	35.99	41.41	43.46
	DeepWalk	36.00	39.60	41.30	42.00
	SpectralClustering	31.06	37.27	40.99	42.62
	EdgeCluster	27.94	31.85	35.00	36.29
	Modularity	27.35	31.77	36.13	38.18
	wvRN	19.51	25.62	31.81	34.28
	Majority	16.51	16.61	16.99	17.26
Macro-F1 (%)	CNN	14.72	24.98	31.92	35.12
	CANE	18.12	23.85	30.04	33.42
	GraphCAN	17.88	23.54	29.87	33.01
	ComE	16.21	22.78	28.98	32.46
	DNNNC	17.55	23.21	29.45	32.83
	SDNE	19.88	24.94	28.11	31.22
	LINE	18.67	24.81	27.91	30.64
	DeepWalk	21.30	25.30	27.60	28.90
	SpectralClustering	19.14	25.97	29.46	31.78
	EdgeCluster	16.16	20.48	23.64	24.92
	Modularity	17.36	20.80	23.41	24.97
	wvRN	6.25	11.64	17.18	19.57
	Majority	2.52	2.52	2.63	2.62

Table 4
Performance on Flickr.

	% Labeled Nodes	1%	3%	6%	9%
Micro-F1 (%)	CNN	25.94	35.88	39.43	44.51
	CANE	23.47	33.49	37.89	42.54
	GraphCAN	23.01	33.10	37.77	42.32
	ComE	22.66	32.68	36.59	41.67
	DNNNC	22.79	32.99	37.14	42.16
	SDNE	23.74	34.76	37.83	41.14
	LINE	23.01	34.44	37.75	40.65
	DeepWalk	32.40	35.90	37.70	38.50
	SpectralClustering	27.43	31.63	33.95	40.14
	EdgeCluster	25.75	29.14	31.53	32.19
	Modularity	22.75	27.30	29.33	29.17
	wvRN	17.70	15.72	19.42	22.51
	Majority	16.34	16.34	16.44	16.67
Macro-F1 (%)	CNN	12.15	20.91	26.46	29.74
	CANE	14.08	19.97	25.73	27.56
	GraphCAN	13.92	19.91	25.63	27.39
	ComE	12.78	19.21	25.11	26.88
	DNNNC	13.78	19.66	25.21	27.01
	SDNE	11.69	19.87	23.29	26.13
	LINE	11.52	19.76	23.01	25.78
	DeepWalk	14.00	19.60	22.90	24.6
	SpectralClustering	13.84	19.44	22.36	23.82
	EdgeCluster	10.52	15.91	18.54	20.78
	Modularity	10.21	15.24	16.64	17.14
	wvRN	1.53	2.91	5.56	8.00
	Majority	0.45	0.45	0.44	0.47

Table 5
Performance on Youtube.

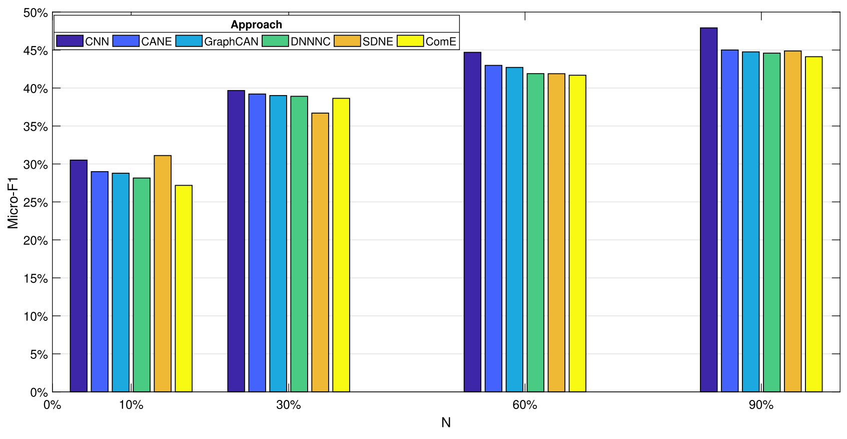
	% Labeled Nodes	1%	3%	6%	9%
Micro-F1 (%)	CNN	33.21	42.18	44.87	47.91
	CANE	36.81	40.07	42.27	45.12
	GraphCAN	36.67	39.99	42.11	44.99
	ComE	36.01	39.24	41.39	43.91
	DNNNC	36.57	39.55	41.89	44.10
	SDNE	34.87	40.24	42.73	44.29
	LINE	34.01	40.11	42.11	43.09
	DeepWalk	37.95	40.08	41.72	42.78
	SpectralClustering	24.41	36.01	39.42	40.21
	EdgeCluster	23.90	35.53	38.63	39.92
	Modularity	23.15	34.98	37.77	39.01
	wvRN	26.79	33.10	37.38	38.68
	Majority	24.90	25.25	25.33	25.38
Macro-F1 (%)	CNN	24.12	34.44	37.85	41.93
	CANE	27.59	33.76	35.24	38.71
	GraphCAN	27.41	33.59	35.12	38.49
	ComE	26.88	32.81	34.66	37.86
	DNNNC	27.12	33.14	34.92	38.06
	SDNE	26.22	33.47	34.88	37.34
	LINE	26.01	33.15	34.76	36.27
	DeepWalk	29.22	33.06	34.66	35.42
	SpectralClustering	20.05	28.77	31.58	32.12
	EdgeCluster	19.48	28.15	30.65	31.45
	Modularity	19.33	27.77	30.11	30.88
	wvRN	13.15	19.66	25.43	28.33
	Majority	6.12	6.21	6.19	6.18

- **SDNE** [43]: This method relies on a deep model using a Laplacian eigenmap.
- **LINE** [44]: A network embedding method based on negative samples and stochastic gradient descent.
- **ComE** [46]: This method relies on node and community embedding for learning graph embeddings.
- **GraphGAN** [48]: A graph representation framework to learn embeddings based on the edge-wised information.
- **CANE** [49]: This framework relies on an adversarial learning framework to jointly learn node representation and identify network communities.
- **DNNNC** [57]: A deep neural network method based on a positive pointwise mutual information (PPMI) matrix for node classification.
- **SpectralClustering**[53]: This method generates a representation in $\mathbb{R}^d$ from the d-smallest eigenvectors of L, the normalized graph Laplacian of G. Utilizing the eigenvectors of L implicitly assumes that graph cuts will be useful for classification.
- **Modularity** [54]: This method generates a representation in $\mathbb{R}^d$ from the top-d eigenvectors of B, the Modularity matrix of G. The eigenvectors of B encode information about modular graph partitions of G, and can be used as features assuming that modular graph partitions will be useful for classification.
- **EdgeCluster** [55]: This method uses k-means clustering to cluster the adjacency matrix of G. It has been shown to perform comparably to the Modularity method, with the added advantage of scaling to graphs which are too large for spectral decomposition.
- **wvRN** [56]: The *weighted-vote Relational Neighbor* is a relational classifier. Given the neighborhood N_i of vertex v_i, wvRN estimates $Pr(y_j|N_j)$ with the (appropriately normalized) weighted mean of its neighbors.
- **Majority**: This naive method simply chooses the most frequent labels in the training set.

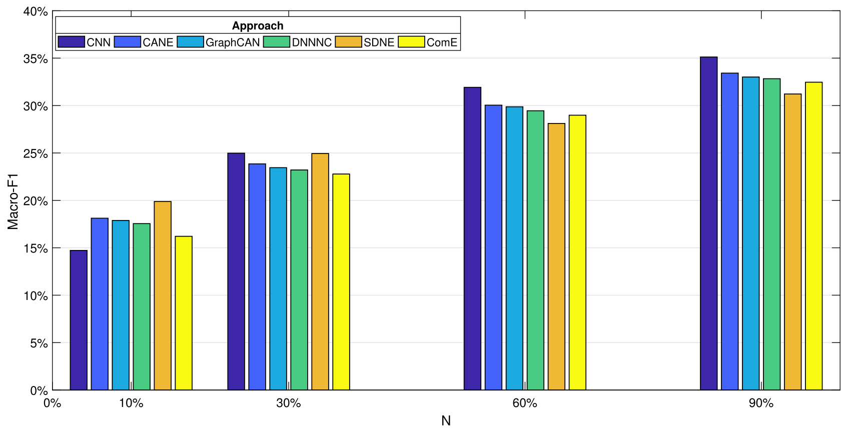
It is important to understand how to set the CNN parameters. First, we analyzed which kind of optimizer is best suitable for our approach. Subsequently, we chose which learning rate to use and how many steps to perform. We leveraged a validation test set, which was obtained by further dividing the training set by $\frac{1}{10}$, and keeping the remaining part as training.

Tables 3–5 show the results obtained by training the CNN with all classes in the various datasets.

Looking at the results, the CNN approach performs poorly when the training set has few instances compared to the test set (cases 10% e 30%). In fact, compared to the baseline algorithms, it has one of the worst performances. However, when the training set has a larger number of instances (cases 60% and



(a) Micro-F1



(b) Macro-F1

Fig. 10. Top-6 approaches in terms of Micro and Macro-F1 on BlogCatalog3 for semi-supervised community detection.

90%), the CNN approach is close to the best case performance. In particular, the ability to process the entire adjacency matrix, thus preserving both local and global knowledge about network structure, provides a richness of information that can be used to improve the performance of the community detection algorithm in contrast to the state-of-the-art approaches which preserve at maximum second order proximity. Furthermore, we model OSN connections as sparse matrices in order to significantly reduce the number of operations that need to be performed, also leveraging different properties of a network, such as topological and context information. This result supports the general insight that CNNs learn better the more instances are used during training.

Finally, Fig. 10, 11, and 12 show the top-6 approaches in terms of Micro and Macro-F1 over the examined datasets.

5.4. Robustness evaluation

We evaluate the effectiveness of the proposed approach on the *email-Eu-core*, varying the number of deleted edges. We focus this evaluation on the *email-Eu-core* dataset because of computational efficiency issues.

Specifically, the training set was evaluated with respect to the loss value according to several parameters: learning rate, number of convolutions and max-Pooling levels, number of kernels, optimizer, and decaying rate.

First, we show how the choice of the optimizer — Gradient Descent or Adam — affects training set performance when varying the number of kernels and the learning rate (0.001, 0.01 and 0.1). Figs. 13, 14, and 15 show how the loss value decreases very slowly when using Gradient Descent with 0.001 up to a value of 3 (after 3000 steps). When choosing a learning rate of 0.1, the loss value instead reaches 0.4 after 1400 steps. With the Adam optimizer, the CNN's loss value of 0.40 in less than 100 steps when the learning rate is set to 0.01 and 0.1, but after 700 steps when the learning rate is set to 0.001.

Finally, we evaluate the robustness of the approach with respect to the network evolution. That is, the goal is to evaluate the accuracy measure when varying the percentage of deleted edges. In particular, given an input matrix A representative of a social graph, we define each entry in the matrix according to the following equation:

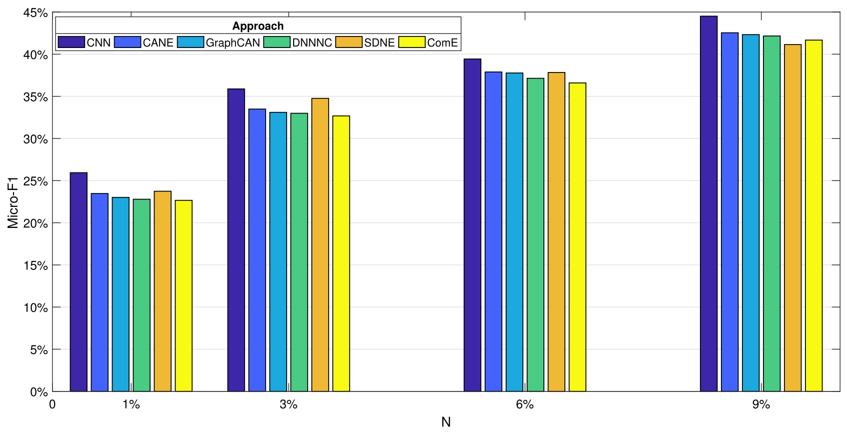
$$A(i,j) = e^{\sigma \cdot (1-s)} \tag{6}$$

where s is the hop count of node n' to node n, with $s_0 \geq s \geq 1$, $\sigma \in (0, 1)$ the attenuation factor and s_0 a user defined hop count threshold. In other words, we verify that node n' is reachable from node n within at least s hops.

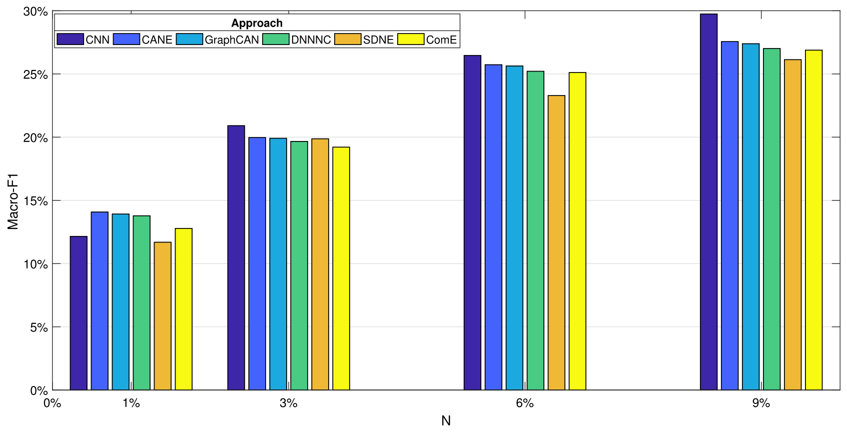
The training set has been evaluated by two convolution and max-pooling levels, 10 kernels, Adam optimizer, and no Decaying with 300 steps varying the s_0 value among 0, 1, 2.

Fig. 16 shows the accuracy of our semi-supervised community detection approach over the test set, with varying hop count $s_0 = 1, 2, 3$. Considering then the matrix filled through entries with $s_0 = 2$, accuracy gets better: for 10% deleted edges the value changes from 78% to 85%, an important increase. However, when comparing the $s_0 = 2$ matrix with the $s_0 = 3$, accuracy does not improve much more. In fact, for lower percentages accuracy over $s_0 = 3$ is worse than $s_0 = 2$.

Considering these results, the best set up to maximize accuracy prediction values is to use a hop count threshold equal to



(a) Micro-F1



(b) Macro-F1

Fig. 11. Top-6 approaches in terms of Micro and Macro-F1 on Flickr for semi-supervised community detection.

2, a right middle way to exploit the advantages of the sparse convolution algorithm.

6. Conclusion and future work

Community detection in OSNs has received more and more attention in the last few years, due to its relevance for a variety of applications, and to the technological challenges that come with it. In particular, the ever growing nature of these systems and the complex ontology of relationships established over them represent important issues to be addressed when developing effective and efficient analysis techniques.

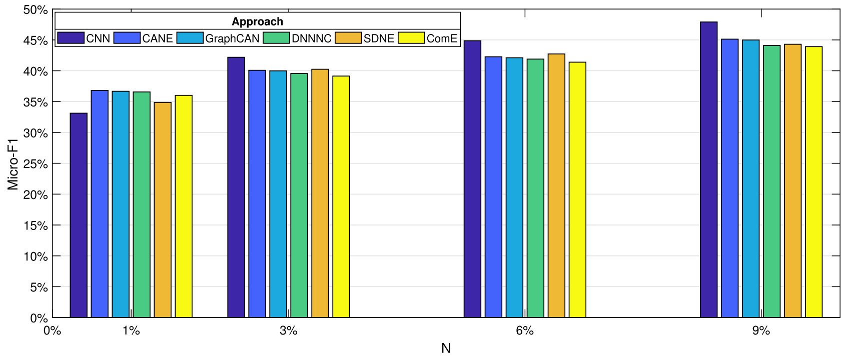
With this in mind, and building on recent deep learning methods, this paper proposed a semi-supervised approach to community detection in large social networks, in order to effectively exploit the topological characteristics of a network's adjacency matrix while addressing main problems and limitation of existing approaches — particularly with respect to the computational requirements associated to large scale datasets.

Building on the insight that the high dimensional adjacency matrices representing social connections in large OSNs are fundamentally sparse, the approach in this paper proposes a modification to the convolutional layer of traditional CNNs, which optimizes computations over sparse matrices (thus highly reducing memory usage) by exclusively considering non-zero values. Importantly, the ability to use the entire adjacency matrix allows us to preserve both local and global knowledge about a network. In turn, and in contrast to other approaches simply preserving second order proximity, our approach is then able to leverage more general information that can be used to improve the performance of community detection. In terms of efficiency, we conducted an extensive evaluation over artificial and real world datasets, showing good performance in terms of running time and accuracy. For instance, our results show a 35% running time decrease with respect to state-of-the-art approaches in the literature.

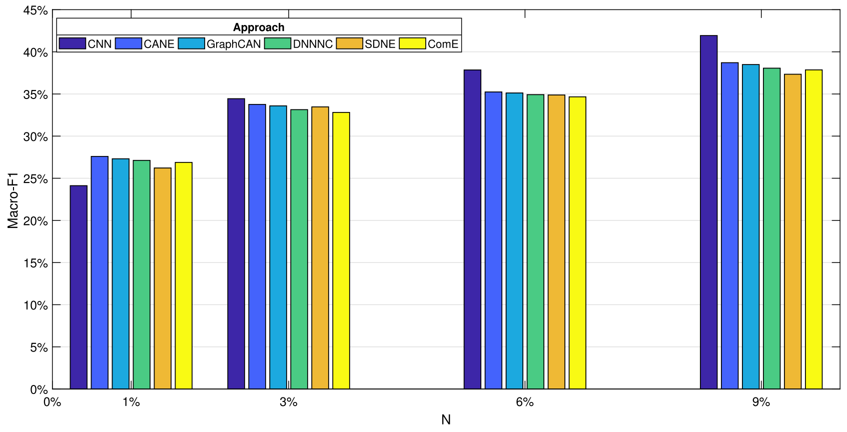
While these results are encouraging, there are a variety of limitations about the proposed solution that should be addressed in future work. In particular, our CNN based approach requires the set up of a complex support infrastructure — mainly based on GPUs. Furthermore, performing parameter tuning (i.e. kernel size, learning rate, number of epoch) at its best can be computationally expensive due to the huge size of the input matrix.

Future work will be devoted to define a new infrastructure based on TensorFlowOnSpark,[2] which enables distributed deep learning, obtaining a faster infrastructure for large scale data processing. Moreover, while we already considered a variety of different datasets, extending the evaluation to datasets representative of distinct OSNs will help further investigations of the ways this approach can be leveraged to improve on different applications of community detection in real world scenarios.

2 https://github.com/yahoo/TensorFlowOnSpark.



(a) Micro-F1



(b) Macro-F1

Fig. 12. Top-6 approaches in terms of Micro and Macro-F1 on Youtube for semi-supervised community detection.

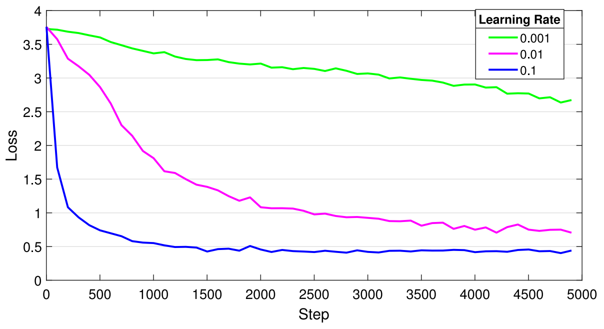


Fig. 13. Training tests using the Gradient Descent optimizer with 3 kernels.

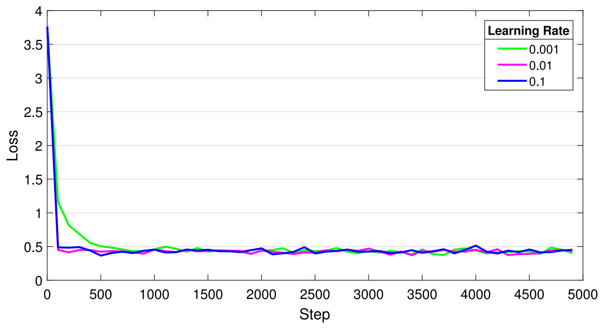


Fig. 15. Training test using Adam optimizer with 10 kernels.

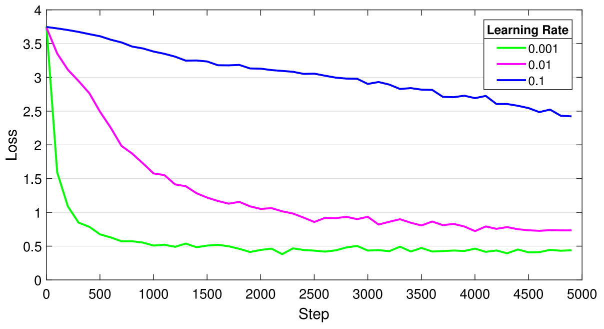


Fig. 14. Training tests using the Gradient Descent optimizer with 10 kernels.

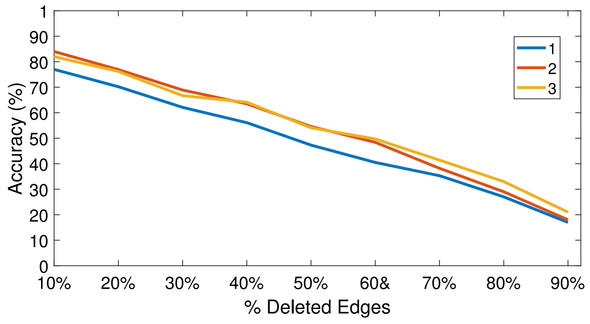


Fig. 16. Accuracy with $s_0 = 1, 2, 3$.

CRediT authorship contribution statement

Aniello De Santo: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing - original draft, Writing - review & editing. **Antonio Galli:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing - original draft, Writing - review & editing. **Vincenzo Moscato:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing - original draft, Writing - review & editing. **Giancarlo Sperlì:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing - original draft, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

All authors approved the version of the manuscript to be published.

References

[1] Statista.com, Social media & user-generated content, 2017, https://www.statista.com/markets/424/topic/540/social-media-user-generated-content.
[2] A. Zareie, A. Sheikhahmadi, M. Jalili, M.S.K. Fasaei, Finding influential nodes in social networks based on neighborhood correlation coefficient, Knowl.-Based Syst. 194 (2020) 105580, http://dx.doi.org/10.1016/j.knosys.2020.105580.
[3] S. Banerjee, M. Jenamani, D.K. Pratihar, Earned benefit maximization in social networks under budget constraint, Expert Syst. Appl. 169 (2021) 114346, http://dx.doi.org/10.1016/j.eswa.2020.114346.
[4] V. Moscato, G. Sperlı, A survey about community detection over on-line social and heterogeneous information networks, Knowl.-Based Syst. 224 (2021) 107112, http://dx.doi.org/10.1016/j.knosys.2021.107112.
[5] X. Ding, J. Zhang, J. Yang, Node-community membership diversifies community structures: An overlapping community detection algorithm based on local expansion and boundary re-checking, Knowl.-Based Syst. 198 (2020) 105935, http://dx.doi.org/10.1016/j.knosys.2020.105935.
[6] Z. Sun, Y. Sun, X. Chang, Q. Wang, X. Yan, Z. Pan, Z. ping Li, Community detection based on the matthew effect, Knowl.-Based Syst. 205 (2020) 106256, http://dx.doi.org/10.1016/j.knosys.2020.106256.
[7] J. Li, T. Cai, K. Deng, X. Wang, T. Sellis, F. Xia, Community-diversified influence maximization in social networks, Inf. Syst. 92 (2020) 101522, http://dx.doi.org/10.1016/j.is.2020.101522.
[8] H. Huang, H. Shen, Z. Meng, H. Chang, H. He, Community-based influence maximization for viral marketing, Appl. Intell. 49 (6) (2019) 2137–2150, http://dx.doi.org/10.1007/s10489-018-1387-8.
[9] J. Zheng, S. Wang, D. Li, B. Zhang, Personalized recommendation based on hierarchical interest overlapping community, Inform. Sci. 479 (2019) 55–75, http://dx.doi.org/10.1016/j.ins.2018.11.054.
[10] E. Yalcin, A. Bilge, Novel automatic group identification approaches for group recommendation, Expert Syst. Appl. 174 (2021) 114709, http://dx.doi.org/10.1016/j.eswa.2021.114709.
[11] P. Bedi, C. Sharma, Community detection in social networks, Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 6 (3) (2016) 115–135, http://dx.doi.org/10.1002/widm.1178.
[12] H. Fani, E. Bagheri, Community detection in social networks, Encycl. Seman. Comput. Robot. Intell. 1 (01) (2017) 1630001, http://dx.doi.org/10.1142/S2425038416300019.
[13] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, P. Spyridonos, Community detection in social media, Data Min. Knowl. Discov. 24 (3) (2012) 515–554, http://dx.doi.org/10.1007/s10618-011-0224-z.
[14] Z. Bu, H.-J. Li, C. Zhang, J. Cao, A. Li, Y. Shi, Graph k-means based on leader identification, dynamic game and opinion dynamics, IEEE Trans. Knowl. Data Eng. (2019) http://dx.doi.org/10.1109/TKDE.2019.2903712.
[15] M. Rezvani, W. Liang, C. Liu, J.X. Yu, Efficient detection of overlapping communities using asymmetric triangle cuts, IEEE Trans. Knowl. Data Eng. 30 (11) (2018) 2093–2105, http://dx.doi.org/10.1109/TKDE.2018.2815554.
[16] X. Zeng, W. Wang, C. Chen, G.G. Yen, A consensus community-based particle swarm optimization for dynamic community detection, IEEE Trans. Cybern. (2019) http://dx.doi.org/10.1109/TCYB.2019.2938895.
[17] X. Ma, L. Gao, X. Yong, L. Fu, Semi-supervised clustering algorithm for community structure detection in complex networks, Physica A 389 (1) (2010) 187–197, http://dx.doi.org/10.1016/j.physa.2009.09.018.
[18] X. Liu, W. Wang, D. He, P. Jiao, D. Jin, C.V. Cannistraci, Semi-supervised community detection based on non-negative matrix factorization with node popularity, Inform. Sci. 381 (2017) 304–321, http://dx.doi.org/10.1016/j.ins.2016.11.028.
[19] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2014, pp. 701–710, http://dx.doi.org/10.1145/2623330.2623732.
[20] A. Grover, J. Leskovec, Node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 855–864, http://dx.doi.org/10.1145/2939672.2939754.
[21] S. Zhu, C. Zhou, S. Pan, X. Zhu, B. Wang, Relation structure-aware heterogeneous graph neural network, in: 2019 IEEE International Conference on Data Mining (ICDM), IEEE, 2019, pp. 1534–1539, http://dx.doi.org/10.1109/ICDM.2019.00203.
[22] N. Gulbahce, S. Lehmann, The art of community detection, BioEssays 30 (10) (2008) 934–938, http://dx.doi.org/10.1002/bies.20820.
[23] M.A. Porter, J.-P. Onnela, P.J. Mucha, Communities in networks, Notices Amer. Math. Soc. 56 (9) (2009) 1082–1097.
[24] B. Yang, D. Liu, J. Liu, Discovering communities from social networks: Methodologies and applications, in: Handbook of Social Network Technologies and Applications, Springer, 2010, pp. 331–346, http://dx.doi.org/10.1007/978-1-4419-7142-5_16.
[25] S. Fortunato, Community detection in graphs, Phys. Rep. 486 (3) (2010) 75–174, http://dx.doi.org/10.1016/j.physrep.2009.11.002.
[26] M.R. Garey, R.L. Graham, Performance bounds on the splitting algorithm for binary testing, Acta Inform. 3 (4) (1974) 347–355.
[27] T. Chakraborty, A. Dalmia, A. Mukherjee, N. Ganguly, Metrics for community analysis: A survey, ACM Comput. Surv. 50 (4) (2017) 54, http://dx.doi.org/10.1145/3091106.
[28] X. Xu, N. Yuruk, Z. Feng, T.A. Schweiger, Scan: a structural clustering algorithm for networks, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2007, pp. 824–833, http://dx.doi.org/10.1145/1281192.1281280.
[29] A. Clauset, M.E. Newman, C. Moore, Finding community structure in very large networks, Phys. Rev. E 70 (6) (2004) 066111, http://dx.doi.org/10.1103/PhysRevE.70.066111.
[30] X. You, Y. Ma, Z. Liu, A three-stage algorithm on community detection in social networks, Knowl.-Based Syst. 187 (2020) 104822, http://dx.doi.org/10.1016/j.knosys.2019.06.030.
[31] V. Moscato, A. Picariello, G. Sperlí, Community detection based on game theory, Eng. Appl. Artif. Intell. 85 (2019) 773–782, http://dx.doi.org/10.1016/j.engappai.2019.08.003.
[32] L. Xiao, W. Wenjun, H. Dongxiao, J. Pengfei, J. Di, C.V. Cannistraci, Semi-supervised community detection based on non-negative matrix factorization with node popularity, Inform. Sci. 381 (2017) 304–321, http://dx.doi.org/10.1016/j.ins.2016.11.028.
[33] W. Wu, S. Kwong, Y. Zhou, Y. Jia, W. Gao, Nonnegative matrix factorization with mixed hypergraph regularization for community detection, Inform. Sci. 435 (2018) 263–281, http://dx.doi.org/10.1016/j.ins.2018.01.008.
[34] M. Rosvall, C.T. Bergstrom, Maps of random walks on complex networks reveal community structure, Proc. Natl. Acad. Sci. 105 (4) (2008) 1118–1123, http://dx.doi.org/10.1073/pnas.0706851105.
[35] F. Mercorio, M. Mezzanzanica, V. Moscato, A. Picariello, G. Sperli, DICO: A graph-DB framework for community detection on big scholarly data, IEEE Trans. Emerg. Top. Comput. (2019) http://dx.doi.org/10.1109/TETC.2019.2952765, 1–1.
[36] Y. Najaflou, K. Bubendorfer, Forming dream teams: A chemistry-oriented approach in social networks, IEEE Trans. Emerg. Top. Comput. (2018) http://dx.doi.org/10.1109/TETC.2018.2869377, 1–1.
[37] F. Lin, W.W. Cohen, Semi-supervised classification of network data using very few labels, in: Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on, IEEE, 2010, pp. 192–199, http://dx.doi.org/10.1109/ASONAM.2010.19.
[38] S. Kumar, W.L. Hamilton, J. Leskovec, D. Jurafsky, Community interaction and conflict on the web, in: Proceedings of the 2018 World Wide Web Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2018, pp. 933–943, http://dx.doi.org/10.1145/3178876.3186141.
[39] L. Yang, X. Cao, D. Jin, X. Wang, D. Meng, A unified semi-supervised community detection framework using latent space graph regularization, IEEE Trans. Cybern. 45 (11) (2015) 2585–2598, http://dx.doi.org/10.1109/TCYB.2014.2377154.

[40] L. Yang, X. Cao, D. He, C. Wang, X. Wang, W. Zhang, Modularity based community detection with deep learning, in: IJCAI, 2016, pp. 2252–2258.
[41] J. Bruna, X. Li, Community detection with graph neural networks, 2017, ArXiv Preprint arXiv:1705.08415.
[42] H. Cai, V.W. Zheng, K.C. Chang, A comprehensive survey of graph embedding: Problems, techniques, and applications, IEEE Trans. Knowl. Data Eng. 30 (9) (2018) 1616–1637, http://dx.doi.org/10.1109/TKDE.2018.2807452.
[43] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1225–1234, http://dx.doi.org/10.1145/2939672.2939753.
[44] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 1067–1077, http://dx.doi.org/10.1145/2736277.2741093.
[45] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, P.S. Yu, Heterogeneous graph attention network, in: The World Wide Web Conference, 2019, pp. 2022–2032, http://dx.doi.org/10.1145/3308558.3313562.
[46] S. Cavallari, V.W. Zheng, H. Cai, K.C.-C. Chang, E. Cambria, Learning community embedding with community detection and node embedding on graphs, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 377–386, http://dx.doi.org/10.1145/3132847.3132925.
[47] B. Li, D. Pi, Learning deep neural networks for node classification, Expert Syst. Appl. 137 (2019) 324–334.
[48] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, M. Guo, Graphgan: Graph representation learning with generative adversarial nets, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32(1), 2018.
[49] J. Wang, J. Cao, W. Li, S. Wang, CANE: community-aware network embedding via adversarial training, Knowl. Inf. Syst. 63 (2) (2021) 411–438, http://dx.doi.org/10.1007/s10115-020-01521-9.
[50] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.
[51] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.
[52] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow.org. https://www.tensorflow.org/.
[53] L. Tang, H. Liu, Leveraging social media networks for classification, Data Min. Knowl. Discov. 23 (3) (2011) 447–478, http://dx.doi.org/10.1007/s10618-010-0210-x.
[54] L. Tang, H. Liu, Relational learning via latent social dimensions, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2009, pp. 817–826, http://dx.doi.org/10.1145/1557019.1557109.
[55] L. Tang, H. Liu, Scalable learning of collective behavior based on sparse social dimensions, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management, ACM, 2009, pp. 1107–1116, http://dx.doi.org/10.1145/1645953.1646094.
[56] S.A. Macskassy, F. Provost, A simple relational classifier, in: Workshop on Multi-Relational Data Mining (MRDM-2003), 2003, p. 64.
[57] B. Li, D. Pi, Learning deep neural networks for node classification, Expert Syst. Appl. 137 (2019) 324–334, http://dx.doi.org/10.1016/j.eswa.2019.07.006.