GEMSEC: Graph Embedding with Self Clustering

Benedek Rozemberczki, Ryan Davies, Rik Sarkar, Charles Sutton
The University of Edinburgh
{benedek.rozemberczki,ryan.davies}@ed.ac.uk,{rsarkar,csutton}@inf.ed.ac.uk

ABSTRACT

Modern graph embedding procedures can efficiently extract features of nodes from graphs with millions of nodes. The features are later used as inputs for downstream predictive tasks. In this paper we propose *GEMSEC* a graph embedding algorithm which learns a clustering of the nodes simultaneously with computing their features. The procedure places nodes in an abstract feature space where the vertex features minimize the negative log likelihood of preserving sampled vertex neighborhoods, while the nodes are clustered into a fixed number of groups in this space. *GEMSEC* is a general extension of earlier work in the domain as it is an augmentation of the core optimization problem of sequence based graph embedding procedures and is agnostic of the neighborhood sampling strategy. We show that *GEMSEC* extracts high quality clusters on real world social networks and is competitive with other community detection algorithms. We demonstrate that the clustering constraint has a positive effect on representation quality and also that our procedure learns to embed and cluster graphs jointly in a robust and scalable manner.

CCS CONCEPTS

•Computing methodologies → Cluster analysis; •Information systems → *Clustering;*

KEYWORDS

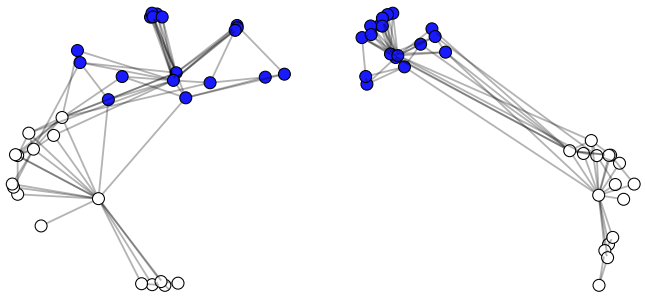
community detection, clustering, graph embedding

1 INTRODUCTION

Finding communities in large networks has been studied extensively [8]. Applications of community detection include the extraction of highly interconnected protein groups from PPI networks [30], identification of research collaborators from citation graphs [2], selection of people with shared interests from social networks [19] or classification of products that are purchased together [26].

A *community* is a set of nodes in the graph which is highly interconnected compared to the graph as a whole. Community detection algorithms use various strategies to find groups of highly interconnected nodes. From a general point of view, community detection is analogous to clustering; one wants to find nodes (data points) which are similar to each other in terms of neighborhoods. Procedures such as the *Louvain* method greedily optimize a quantity called modularity representing the quality of extracted clusters [3, 6]. Other algorithms start random processes and find communities based on the diffusion properties. *Walktrap* extracts closely knit neighborhoods with random walks [20] while *Label Propagation* uses labels randomly propagated through the network to identify communities [10].

In a similar manner, sequence based graph embedding methods like *DeepWalk* and *Node2Vec* use random processes to generate representations of nodes in the graph [11, 21, 28]. Graph embedding procedures generally map nodes to a low dimensional abstract space where distances among nodes are approximately preserved. Besides distances, the neighborhood structure is also preserved to a certain extent. This means that nodes with overlapping neighborhoods are positioned close to each other in the low dimensional space. The learned representations are useful for machine learning tasks such as the labeling of the nodes, regression, edge prediction and graph visualization [9].



(a) DeepWalk **(b) GEMSEC**

Figure 1: Zacahry's Karate club visualized using Deepwalk and GEMSEC embeddings. White nodes correspond to the instructor's community and blue ones to the president's community.

Our contributions. We propose *GEMSEC* – a graph embedding procedure which learns an embedding of nodes and clusters the nodes at the same time. Sequence based graph embedding procedures create d dimensional feature representations of nodes in an abstract latent space that represent the likelihood of observing similar sampled neighborhoods of the nodes. This problem can be reformulated as minimizing the negative log likelihood of observed neighborhood samples in a probabilistic model on graphs. We extend this reformulated objective function by adding clustering cost to the negative log likelihood and solve the resulting optimization problem with a variant of mini-batch gradient descent [7]. Simply, our approach creates a d dimensional representation for each node and C cluster centers in this d dimensional Euclidean space. Our model is agnostic of the way neighborhoods of vertices are being sampled. In our experiments we specifically settled on using first order truncated random walks.

The fundamental idea in our work is that sequence based graph embedding procedures create similar representations for nodes which have similar sampled neighborhoods. Nodes with similar neighborhoods are expected to be in the same community. This means that clustering the representations can reveal the community structure. We directly exploit this feature by adding a cost element

which enforces the created representation to be clustered. Looking at Figure 1 the intuition behind our algorithm can be easily understood. By enforcing nodes to be clustered, *GEMSEC* reveals the natural community structure in the graph.

In addition we demonstrate that the objective function of sequence based graph embedding procedures can be extended with a smoothness regularization term. This procedure can additionally force nodes with high neighborhood overlaps to have similar representations.

The experimental evaluation of *GEMSEC* primarily focuses on non-overlapping community detection. We demonstrate that *GEMSEC* outperforms graph embedding procedures [21] and community detection methods [6, 10, 20] on a variety of real world social networks collected from Facebook when it comes to clustering. In terms of the clusterings' modularity, the performance advantage of our proposed method over the baseline community detection procedures varies between 0.24% and 10.74%. We also highlight with Deezer music streaming data that the clustering constraint has a positive effect on the representation quality of social network features for music genre recommendation. In terms of micro-averaged F1 score, the predictive performance increase compared to *DeepWalk* is between 3.74% and 8.79% in predicting genres liked by users using the embeddings. Our results support that the clustering performance of *GEMSEC* is robust to hyperparameter changes and also to perturbation. Finally, we show that the runtime complexity of our method is linear in the number of nodes.

Main contributions of our work are:

(1) We present *GEMSEC*, a sequence based graph embedding model which learns a graph embedding and a clustering of vertices in the embedding space jointly.
(2) We introduce a procedure to train *GEMSEC* and demonstrate that its runtime scales linearly with the number of vertices.
(3) We enhance *GEMSEC* with smoothness regularization in order to enforce nodes with high neighborhood overlap to have more similar representations.
(4) We show strong clustering and embedding capabilities of *GEMSEC* on social networks extracted from Facebook and Deezer.

The remainder of the paper is structured as follows. In Section 2 we overview the related literature. We discuss the graph clustering algorithm and its variants in Section 3. The evaluation of the introduced methods using real world social network data is carried out in Section 4. We look at the cluster quality obtained with our procedure and compare the clustering performance to other community detection methods. We show that the introduction of the clustering constraint increases representation quality on the task of multi-label node classification. We demonstrate the scalability of our algorithm, carry out a sensitivity analysis and highlight its robustness to perturbation. The paper concludes with Section 5 where we review our main findings and outline possible extensions of our work. The datasets that we collected and used in the experiments will be made available on https://snap.stanford.edu/. A high performance reference implementation of *GEMSEC* can be accessed on https://github.com/benedekrozemberczki.

2 RELATED WORK

Recent advances in graph embedding procedures have made possible the effective and scalable unsupervised feature learning for large real world graphs [11, 21, 28]. Features extracted with these *sequence based graph embedding* procedures can be used for predicting social network users' missing age [23], the category of scientific papers in citation networks [15] and the function of proteins in protein-protein interaction networks [11]. Besides supervised learning tasks on nodes the extracted features can be used for graph visualization [9], edge prediction [11, 24], community detection [9] and structural role identification [27].

Sequence based graph embedding procedures were inspired by distributed word representations, specifically by the Skip-Gram model [16, 17]. In this model representations of words are learned to minimize the negative log likelihood of observing words in the neighborhood (context) of a specific word. These models use sentences in order to generate the contexts which one can think of as linear sequences of strings. The minimization problem itself is solved with stochastic gradient descent and made tractable by the use of negative sampling [12]. In a similar manner, graph embedding models first sample sequences of vertices and extract noisy proximity statistics from these sequences within a fixed context. Co-occurrence statistics are encoded with a single hidden layer neural network and node specific weights in the input layer are used as vertex representations.

Research in sequence based graph embedding literature mainly considers the definition of the sampling strategy that is used to obtain vertex sequences. The most basic sampling strategy is the use of truncated random walks as used in *Deepwalk* [21]. More involved sequence sampling methods include the use of second-order random walks [11], the introduction of skips in random walks [22] and branching processes [24]. These more sophisticated synthetic data generation models encode the structural role of nodes [27], to get a representation that is in line with the multi level structure of the graph and consequently to improve the predictive performance on downstream machine learning tasks. There is also a number of works focused on creating embeddings for specific types of graphs. Specifically, there is work looking at graphs with signed edges [35] and rooted subgraphs [18]. Another straightforward contribution to this area of research was the inclusion of generic vertex features in the creation of representations [34].

Besides sequence based graph embedding procedures other neural models were developed specifically to create representations of graphs for downstream machine learning tasks [4, 31]. Unlike sequence based embedding algorithms these models encode directly the adjacency matrix and other matrix representations of the graph. A major development in this direction was the introduction of scalable graph convolutional networks which besides the topology used generic features of vertices to create representations of nodes [15]. Representations created with graph convolutional models have a shortcoming similar to the majority of sequence based graph embedding procedures. They are not inductive, meaning that the learned representations cannot be transferred to other graphs. However, combined with branching process sampling, scalable inductive methods have been developed recently [14]. The possibility of combining self clustering embeddings with these methods to

create community detection procedures that use node features remains to be investigated.

There has been previous work on using learned representations for community detection, but existing research is limited to clustering the nodes after creating an embedding [5, 29]. Other methods aim to create representations and clusterings jointly that are consistent with precomputed clusterings [25, 33]. Our approach on the other hand, takes clustering quality directly account while performing the embedding optimization, thus producing results faster and a more natural embedding with clusters.

3 GRAPH EMBEDDING WITH SELF CLUSTERING

In this section we set up the optimization problem of jointly learning graph embedding and clustering. First, we focus on the optimization problem that allows for learning the embedding solely. Now let $G = (V, E)$ be the graph of interest where V is the set of vertices and E is the set of edges. The graph embedding is a mapping $f : V \rightarrow \mathbb{R}^d$ where d is the dimensionality of the embedding space. For each node $v \in V$ we create a d dimensional representation, essentially the embedding f is a $|V| \times d$ real valued matrix. In the sequence based embedding, sequences of neighboring nodes are sampled from the graph. Within a sequence, a node v occurs in the context of a window ω which we call neighbors of v in the sequence. Given a strategy S to sample sequences, we refer to the collection of windows containing v as $N_S(v)$. Earlier works have proposed random walks, second order random walks or branching processes to obtain $N_S(v)$. In our experiments presented later we will use unweighted first order random walks for sampling.

Our goal is to minimize the negative log likelihood of observing these neighborhoods of source nodes conditional on the feature vectors that describe the position of nodes in the embedding space. The representation vector specific to node v is $f(v)$. The optimization problem of interest is given by:

$$\min_{f} \quad \sum_{v \in V} -\log P(N_S(v)|f(v)). \tag{1}$$

This optimization is subject to two standard assumptions [11]. First, we assume that $P(N_S(v)|f(v))$ can be factorized in line with *conditional independence* with respect to $f(v)$. The consequence of this assumption is:

$$P(N_S(v)|f(v)) = \prod_{n_i \in N_S(v)} P(n_i \mid f(v)). \tag{2}$$

Second, we demand *symmetry in the feature space*, meaning that source and neighboring nodes have a symmetric effect on each other in the embedding space. We achieve this by using a softmax function on the pairwise dot products of node representations with $f(v)$ to get $P(n_i|f(v))$. This is expressed by

$$P(n_i|f(v)) = \frac{\exp(f(n_i) \cdot f(v))}{\sum_{u \in V} \exp(f(u) \cdot (f(v))}. \tag{3}$$

Using Equations (2) and (3) we can reformulate the optimization problem as

$$\min_{f} \quad \sum_{v \in V} \left[\ln \left(\sum_{u \in V} \exp(f(v) \cdot f(u)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right]. \tag{4}$$

The partition function in Equation (4) enforces nodes to be embedded in a low volume space around the origin, while the second term forces nodes with similar sampled neighborhoods to have similar representations, but farther out from the origin.

3.1 Learning to Cluster

We have formulated the sequence based graph embedding problem as a minimization task, and now we can extend it with a proper clustering cost and define the *GEMSEC* model. As we discussed in the introduction, a sequence based graph embedding procedure will place nodes with similar neighborhoods close to each other in the created latent space. Formally this means that $N_S(v) \sim N_S(u)$ implies that $f(v) \sim f(u)$ conversely we can say that $f(v) \sim f(u)$ implies that $N_S(v) \sim N_S(u)$.

The optimization problem described by Equation (4) can be augmented with a k-means like cost function which describes the clustering cost specifically in the embedding space. This augmented optimization problem is described by

$$\min_{f,\mu} \quad \underbrace{\sum_{v \in V} \left[\ln \left(\sum_{u \in V} \exp(f(v) \cdot f(u)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right]}_{\text{Embedding cost}} + \underbrace{\gamma \cdot \sum_{v \in V} \min_{c \in C} \|f(v) - \mu_c\|_2^2}_{\text{Clustering cost}}. \tag{5}$$

In Equation (5) we have C cluster means – the c^{th} cluster mean is denoted by μ_c. Each of these cluster centers is a d-dimensional vector in the embedding space. For each node we calculate the distance from the cluster centers and we take the smallest of these distances. The weight coefficient of the clustering cost is given by the hyperparameter γ. Cluster means themselves are trainable parameters just as the embedding vectors are. Evaluating the partition function in the proposed objective function for all of the source nodes has a $O(|V|^2)$ runtime complexity. Because of this, we have to approximate the partition function term with negative sampling which is a form of noise contrastive estimation [12, 16].

The objective function has insightful gradients with respect to the node representation vectors and clusters centers. Assuming that all cluster centers are distinct, the minimum function in the clustering cost is differentiable. As a result we can obtain the gradients for node representations and cluster centers. Understanding these gradients is essential as we will use a first-order gradient-based optimizer to solve the problem proposed in Equation 5.

Specifically, the gradient of the objective function $\mathcal{L}$ with respect to the representation of node $v^* \in V$ is described by Equation (6) if μ_c is the closest cluster center to $f(v^*)$.

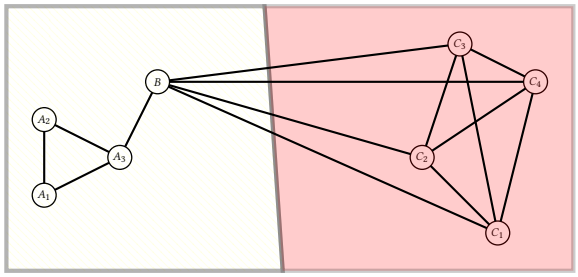
$$\frac{\partial \mathcal{L}}{\partial f(v^*)} = \underbrace{\frac{\sum_{u \in V} \exp(f(v^*) \cdot f(u)) \cdot f(u)}{\sum_{u \in V} \exp(f(v^*) \cdot f(u))}}_{\text{Partition function gradient}} - \underbrace{\sum_{n_i \in N_S(v^*)} f(n_i)}_{\text{Neighbor direction}} + \underbrace{\gamma \cdot 2 \cdot (f(v^*) - \mu_c)}_{\text{Closest cluster direction}} \tag{6}$$

First, the gradient of the partition function pulls the representation of v^* towards the origin. The second term moves the representation of v^* closer to the representations of its neighbors in the embedding space while the third term moves the node closer to the closest cluster center. First, we have to note that if we set a high γ value the third term dominates the gradient. This will cause the node to gravitate towards the closest cluster center which might not contain the neighbors of v^*. This phenomenon can be understood by looking at Subfigure 2a. If node B is initially captured in the cluster on the left hand side it will never end up in a cluster with its neighbors on the right hand side. Second, only the position of the closest cluster center affects the representation of $f(v^*)$.

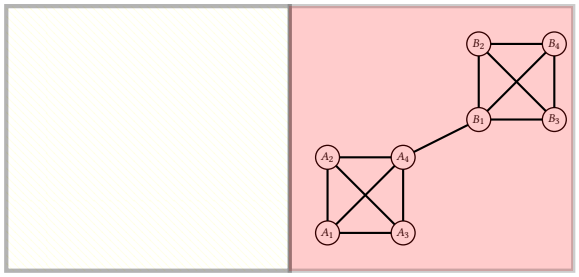
If the set of nodes that belong to cluster center c is V_c the gradient of the objective function with respect to μ_c is described by

$$\frac{\partial \mathcal{L}}{\partial \mu_c} = -\gamma \cdot 2 \sum_{v \in V_c} (f(v) - \mu_c). \tag{7}$$

First, looking at Equation 7 we see that the gradient moves the cluster center by the sum of coordinates of nodes in the embedding space that belong to cluster c. Second, if a cluster ends up empty it will not be updated as elements of the gradient would be zero. Because of this, the initialization of the cluster centers has to be chosen carefully. A wrong initialization just like the one with an empty cluster on Subfigure 2b can affect clustering performance considerably.



(a) Node capture due to high cluster cost coefficient



(b) Empty cluster initialization

Figure 2: Potential issues with cluster cost weighting and cluster initialization. Different background colors denote different sides of the clustering decision boundary.

We want to create an embedding of nodes that preserves neighborhoods and at the same time we want the nodes to cluster in the embedding space around the cluster centers. In order to do so we have to set the cost coefficient of clustering such way that node representations are not captured by the closest cluster centers right after initialization. The main difference between this procedure and simply clustering the nodes after embedding them is the fact that the learned node representations can adapt to the clustering constraint.

We propose an efficient learning method to create *GEMSEC* embeddings which is described with pseudo-code by Algorithm 1. The main idea behind our procedure is the following. If initially the weight of clustering is too high, we will not be able to create meaningful representations and cluster centers at the same time. To resolve this, we anneal the weight coefficient of the clustering cost from a γ starting value to one. This way the learned representations can adapt slowly to the cluster centers.

Data: $\mathcal{G} = (V, E)$ – Graph to be embedded.
N – Number of sequence samples per node.
l – Length of sequences.
ω – Context size.
d – Number of embedding dimensions.
C – Number of clusters.
k – Number of noise samples.
γ – Initial clustering weight coefficient.
η – Learning rate.
Result: $f(v)$, where $v \in V$
μ_c, where $c \in C$

1 Model ← **Initialize Model**($|V|$, d, C)
2 **for** n in *1:N* **do**
3 $\widehat{V}$ ← **Shuffle**(V)
4 **for** v in $\widehat{V}$ **do**
5 γ ← **Update Cluster Cost Weight**(γ)
6 η ← **Update Learning Rate**(η)
7 Sequence ← **Sample Nodes**($\mathcal{G}$, v, l)
8 Features ← **Feature Extraction**(Sequence, ω)
9 **Update Model Weights**(Model, Features, γ, η, k)
10 **end**
11 **end**

Algorithm 1: GEMSEC training procedure

To train an embedding we do the following. Based on the number of vertices, embedding dimensions and clusters, we initialize the weights of our model. After this we do N sampling repetitions in order to generate vertex sequences from every source node. Before starting a sampling epoch we shuffle the set of vertices. We iterate through the shuffled vertex set node by node. We increase the clustering cost coefficient γ (line 5) and decrease the learning rate η (line 6). From the source node with our sampling strategy S we sample a vertex sequence with length l (line 7) and extract features using the context size ω (line 8). Using the extracted features, current learning rate and clustering cost coefficient we update the model weights using the gradients and our chosen optimizer (line 9). In our implementation we utilized a variant of stochastic gradient descent, specifically we used the Adam optimizer [7]. As we said earlier to make the optimization problem tractable we approximate the first cost term with noise contrastive estimation, for each positive sample we draw k noise samples. The clustering cost coefficient is exponentially and the learning rate is linearly annealed. The clustering cost coefficient is being increased whereas the learning rate is being decreased. All in all, the runtime complexity of this procedure is $O((\omega \cdot k + C) \cdot l \cdot d \cdot |V| \cdot N)$ while *DeepWalk* with

noise contrastive estimation has a $O(\omega \cdot k \cdot l \cdot d \cdot |V| \cdot N)$ runtime complexity.

3.2 Smoothness Regularization

The graph embedding objective function can be augmented further by adding a smoothness regularization cost element which encourages learned representations to be similar if two sampled nodes share an edge [13, 32]. This way we could increase the weight on having well represented strong first order relationships. We could add a smoothness regularization both to Equations (4) and (5). If Λ is added to the objective function described by Equation (4) we get the smoothed graph embedding model. Later we reference this model as *Smooth DeepWalk*. We also add the regularization to the objective function of *GEMSEC* doing so we define *Smooth GEMSEC*.

$$\Lambda = \lambda \cdot \sum_{(v,u)\in E_S} w_{(v,u)} \cdot \|f(v) - f(u)\|_2 \tag{8}$$

In Equation (8) each node pair (v, u) has a regularization weight $w_{(v,u)}$. The weight can be arbitrarily chosen, later we discuss several different alternatives. The hyperparameter λ describes the regularization coefficient. If λ is high, distant representations of nodes with an edge between them are penalized heavily. The regularization term is being summed over the edges obtained with the sampling strategy.

Regularization Weight	Definition
Unit	1
Neighborhood overlap	$\lvert\mathcal{B}(v) \cap \mathcal{B}(u)\rvert$
Jaccards's Coefficient	$\frac{\lvert\mathcal{B}(v) \cap \mathcal{B}(u)\rvert}{\lvert\mathcal{B}(v) \cup \mathcal{B}(u)\rvert}$
Minimum normalized overlap	$\frac{\lvert\mathcal{B}(v) \cap \mathcal{B}(u)\rvert}{\min(\lvert\mathcal{B}(v)\rvert, \lvert\mathcal{B}(u)\rvert)}$

Table 1: Smoothness regularization weights for edge (v, u) with first order neighbor sets $\mathcal{B}(v)$ and $\mathcal{B}(u)$.

The training procedure described by Algorithm 1 needs to be modified slightly to accommodate the introduction of the smoothness regularization term. First, we need to set a regularization coefficient. Using the sampled sequence of vertices we extract the edges on which we want to enforce the regularization term. For each sampled edge (v, u) we have to obtain the weight $w_{v,u}$. The optimizer besides the regularization coefficient also needs this list of edges and weights for doing an update of model parameters.

In our work we consider different types of regularization weights that we use to obtain smooth representations. Their definitions are summarized in Table 1. Each of these regularization weights penalizes the difference of node representations with a different strategy.

- *Unit:* On each edge we have the same $w_{v,u} = 1$ weight.
- *Neighborhood overlap:* If two nodes have a large overlapping neighborhood they should have similar representations and this weight increases linearly with the number of shared neighbors. This weighting strategy is biased towards pairs of nodes with high degree.
- *Jaccards's Coefficient:* The neighborhood overlap is normalized by the union of the neighborhood sets in order to down weight nodes with large neighborhood sets. Edges that are between nodes with no common neighbor have zero weight.
- *Minimum normalized overlap:* This weighting strategy gives high weight to an edge when one of the nodes has neighbors that are all neighbors of the other node.

Source	Dataset	|V|	Density	Transitivity
Facebook	Politicians	5,908	0.0024	0.3011
	Companies	14,113	0.0005	0.1532
	Athletes	13,866	0.0009	0.1292
	News Sites	27,917	0.0005	0.1140
	Public Figures	11,565	0.0010	0.1666
	Artists	50,515	0.0006	0.1140
	Government	7,057	0.0036	0.2238
	TV Shows	3,892	0.0023	0.5906
Deezer	Croatia	54,573	0.0004	0.1146
	Hungary	47,538	0.0002	0.0929
	Romania	41,773	0.0001	0.0752

Table 2: Statistics of the datasets used in the paper for evaluation.

4 EXPERIMENTAL EVALUATION

In this section we evaluate whether the cluster quality obtained by *GEMSEC* is competitive with other community detection procedures. We investigate the scalability and robustness of our method. We also explore how clustering affects the performance on downstream predictive tasks.

Standard parameter settings. In order to help with reproducibility we have a standard parameter setting which we use in our experiments to create embeddings. We emphasize when we deviate from these settings. Unweighted truncated random walks with length 80 are used as a sampling strategy and we do 5 random walks per source node. From the random walks, features are extracted with a window size of 5. The embedding has 16 dimensions and it is assumed that we need to extract 20 cluster centers. We did a parameter sweep over a number of hyperparameters to obtain the highest average modularity on a given dataset. Initial learning rate values are chosen from $\{10^{-2}, 5 \cdot 10^{-3}, 10^{-3}\}$, in a similar manner the final learning rate is chosen from $\{10^{-3}, 5 \cdot 10^{-4}, 10^{-4}\}$. Noise contrastive estimation uses 10 negative examples per positive ones. The initial clustering cost coefficient is chosen from $\{10^{-1}, 10^{-2}, 10^{-3}\}$ and it is annealed to 1. Coefficient of the regularization term is 0.0625. Lastly, Jaccard's coefficient is used as a smoothness penalty weight as it consistently gives high quality results.

Datasets used. For the evaluation of *GEMSEC* real-world social network datasets are used that we collected from public API's specifically for this work. As you can see in Table 2 these social networks have a variety of size, density and level of clustering. We used graphs from two sources:

- *Facebook page networks:* These graphs represent mutual like networks among verified Facebook pages – the types of sites included TV shows, politicians, athletes and artists among others.

	Facebook Verified Site Networks							
	Politicians	**Companies**	**Athletes**	**News Sites**	**Public Figures**	**Artists**	**Government**	**TV Shows**
Overlap Factorization	0.810 (±0.008)	0.553 (±0.010)	0.601 (±0.020)	0.471 (±0.016)	0.551 (±0.01)	0.474 (±0.018)	0.608 (±0.024)	0.786 (±0.008)
Walktrap	0.841 (±0.023)	0.639 (±0.016)	0.670 (±0.021)	0.514 (±0.023)	0.628 (±0.023)	0.554 (±0.026)	0.675 (±0.043)	0.790 (±0.036)
Fast greedy	0.819 (±0.008)	0.665 (±0.014)	0.605 (±0.026)	0.531 (±0.020)	0.630 (±0.011)	0.464 (±0.023)	0.615 (±0.046)	0.835 (±0.006)
Label Propagation	0.826 (±0.009)	0.647 (±0.075)	0.647 (±0.094)	0.243 (±0.159)	0.612 (±0.027)	0.393 (±0.018)	0.659 (±0.041)	0.839 (±0.004)
DeepWalk	0.840 (±0.017)	0.637 (±0.012)	0.649 (±0.012)	0.481 (±0.022)	0.631 (±0.011)	0.508 (±0.029)	0.703 (±0.010)	0.831 (±0.004)
Smooth DeepWalk	0.849 (±0.017)	0.667 (±0.007)	0.669 (±0.007)	0.541 (±0.006)	0.643 (±0.008)	0.523 (±0.020)	0.707 (±0.008)	0.835 (±0.008)
GEMSEC	0.851 (±0.009)	0.650 (±0.013)	0.674 (±0.009)	0.536 (±0.011)	0.636 (±0.014)	0.528 (±0.020)	0.705 (±0.020)	0.833 (±0.010)
Smooth GEMSEC	**0.855** (±0.006)	**0.683** (±0.009)	**0.694** (±0.009)	**0.588** (±0.009)	**0.649** (±0.007)	**0.559** (±0.011)	**0.716** (±0.008)	**0.841** (±0.004)

Table 3: Mean modularity of clusterings on the Facebook datasets. Each embedding experiment was repeated ten times. Errors in the parentheses correspond to two standard deviations. In terms of modularity *GEMSEC* outperforms the baselines on most of the datasets. *Smooth GEMSEC* consistently does better than the baselines and *DeepWalk*.

- *Deezer user-user friendship networks:* We collected friendship networks from the music streaming site Deezer and we included 3 European countries (Croatia, Hungary and Romania). For each user we curated the list of genres loved based on the songs liked by the user.

4.1 Cluster Quality

Using the Facebook page networks we evaluate the clustering performance. We use the standard parameter settings of our model to do clustering. We did a grid search over the standard learning rate and clustering cost coefficient values. Results obtained by k-means clustering *DeepWalk* embedding features and benchmarks using baseline community detection methods are also included. The cluster quality is evaluated by modularity – we assume that a node belongs to a single community. Our results are summarized in Table 3. Numbers report the mean modularity based on 10 randomly initialized experimental repetitions. Errors in parentheses correspond to two standard deviations. Baseline community detection methods used for comparison are:

(1) *Overlap Factorization* [1]: We factorize the neighborhood overlap matrix to create continuous features. We create a 16 dimension factorization and cluster the factors with k-means clustering to create 20 communities.
(2) *Walktrap* [20]: Initiates random walks from every node and clusters vertices hierarchically based on the obtained neighborhoods. From each node we do truncated random walks with length 5.
(3) *Fast Greedy* [6]: Optimizes modularity greedily by clustering vertices with a hierarchical procedure.
(4) *Label Propagation* [10]: Creates unique labels for nodes that are propagated in the network using a random order majority rule based voting. We did label propagation that is randomly initialized and unweighted.

First, we see that *Smooth GEMSEC* consistently outperforms the embedding based methods and the benchmark community detection methods. Our procedure's performance advantage over the community detection benchmarks is the highest on the News Sites dataset the clustering's modularity is 10.73% higher and it is the lowest on the TV shows dataset with an advantage of 0.24%. Second, the basic *GEMSEC* method marginally outperforms *DeepWalk* on every dataset. Third, the use of smoothness regularization has sometimes non-significant, but definitely positive effect on the clustering performance of both *Deepwalk* and *GEMSEC*. Lastly, we assume that a careful tuning of hyperparameters could increase the clustering performance further. For example we fixed the cluster number at 20, but it is fairly plausible that other values would give considerably better results on some of these graphs.

4.2 Music Genre Recommendation

Graph embeddings are mainly used for extracting features of nodes for downstream predictive tasks. As we modified the graph embedding objective it is fair to assume that the performance on predictive problems might change due to the reformulation of the optimization problem itself. We might expect that accuracy increased or dropped by enforcing the clustering constraint. In order to investigate this we use social networks of Deezer users collected from European countries. We predict the genres of music liked by people using the embeddings. The number of distinct genres that users can like is 84 in each dataset.

The exact experimental setup was as follows. Each graph was embedded with the standard parameter settings that gave the highest modularity. *DeepWalk* embeddings used initial and final learning rate valuers such as 10^{-3} and 10^{-4}. For *GEMSEC* embeddings we modified the initial and final learning rates to be 10^{-3} and $5 \cdot 10^{-4}$ respectively and the initial cost coefficient of the clustering was 0.1. We used logistic regression with ℓ_2 regularization to predict each of the labels and 90% of the nodes were randomly selected for training. We evaluated the performance on the remaining users. The reported numbers in Table 4 are mean micro, macro and weighted average F1 scores calculated from 10 experimental repetitions.

When performance is evaluated by micro-averaged F1 score we see that *Smooth GEMSEC* significantly outperforms *DeepWalk* on all three datasets. Precisely this performance advantage varies between 3.73% and 8.79%. We also see that the basic *GEMSEC* is able to outperform *DeepWalk* on the Hungarian and Romanian Deezer datasets when performance is measured by micro averaged F1 score. Moreover, we see that based on this metric the performance of *Smooth DeepWalk* on this downstream task is as good as

	Deezer Social Networks								
	Croatia			Hungary			Romania		
	Micro	Macro	Weighted	Micro	Macro	Weighted	Micro	Macro	Weighted
DeepWalk	0.321 (±0.006)	0.026 (±0.002)	0.207 (±0.004)	0.361 (±0.004)	0.029 (±0.002)	0.228 (±0.002)	0.307 (±0.008)	0.023 (±0.002)	0.186 (±0.006)
Smooth DeepWalk	0.329 (±0.006)	**0.028** (±0.002)	**0.215** (±0.006)	0.375 (±0.006)	0.032 (±0.002)	0.244 (±0.004)	0.321 (±0.008)	0.026 (±0.002)	0.204 (±0.006)
GEMSEC	0.328 (±0.006)	0.027 (±0.002)	0.212 (±0.004)	0.377 (±0.004)	0.032 (±0.002)	0.244 (±0.004)	0.332 (±0.008)	0.028 (±0.002)	0.213 (±0.006)
Smooth GEMSEC	**0.333** (±0.006)	**0.028** (±0.002)	**0.215** (±0.004)	**0.379** (±0.006)	**0.034** (±0.002)	**0.250** (±0.004)	**0.334** (±0.008)	**0.029** (±0.002)	**0.215** (±0.006)

Table 4: Multi-label node classification performance of the embedding extracted features on the Deezer genre likes datasets. Performance is measured by average F1 score values. Models were trained on 90% of the data and evaluated on the remaining 10%. Errors in the parentheses correspond to two standard deviations. *GEMSEC* models consistently have good performance.

the performance of *GEMSEC* variants. The results obtained with micro and weighted average F1 score also underpin these findings.

These results are quite interesting considering the fact that the number of naturally existing communities in these graphs might be quite different from the cluster number that we enforced in our experiment. In addition, we have evidence that introducing the clustering cost and smoothness regularization does not affect performance on the downstream predictive task adversely. On the contrary, it can increase the predictive accuracy significantly.

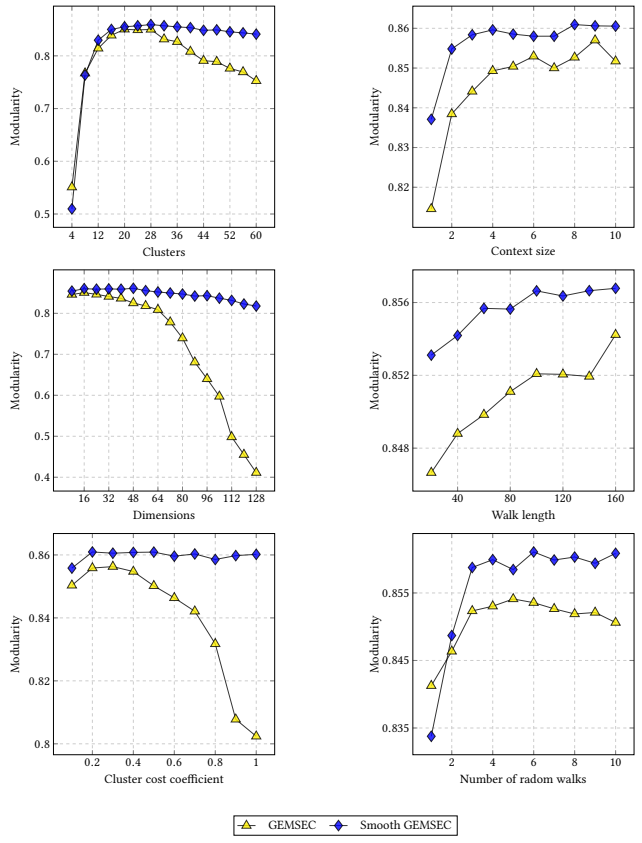


Figure 3: Sensitivity of cluster quality to parameter changes measured by modularity. Smoothness regularization increases the robustness of the self-clustering embedding.

4.3 Sensitivity Analysis

The formulation of *GEMSEC* involves the definition of a number of hyperparameters which affect the representation and henceforth the cluster quality. In the following experiments we test how the manipulation of hyperparameters affects the clustering performance of the introduced models. We embed the Politicians Facebook graph with the standard parameter settings while the initial and final learning rates are set to be 10^{-2} and $5 \cdot 10^{-3}$ respectively, the clustering cost coefficient is 0.1 and we perturb certain hyperparameters. In Figure 3 each data point represents the mean modularity calculated from 10 experimental repetitions. We test the sensitivity of clustering performance to cluster center number, context size, dimension number, random walk length, clustering cost coefficient and initiated random walks per source node.

Based on the experimental results we can make two main observations. First, *GEMSEC* models give high quality clusterings for a wide range of parameter settings. Second, introducing smoothness regularization makes *GEMSEC* more robust to hyperaparameter changes when performance is evaluated by cluster quality. Specifically, we observe that increasing the cluster number above 30 decreases slightly the average modularity. In case of the non smooth model the decrease in cluster quality is quite considerable.

We conclude that increasing the context size, the length of truncated random walks and the number of random walks per source node above a certain threshold has only marginal effect on the community detection performance. Interestingly, we also have empirical evidence for the node capture phenomenon — if the clustering cost coefficient is too high vanilla *GEMSEC* has a poor clustering performance. Finally, there is strong evidence that both *GEMSEC* and *Smooth GEMSEC* perform poorly when the number of dimensions used to create the embedding is high.

4.4 Smoothness Regularization Strategies

As we have seen the introduction of smoothness regularization can boost the clustering performance considerably. In Subsection 3.2 we defined different strategies for weighting edges in the smoothness regularization cost. It is presumed that these strategies work well with different values of the regularization coefficient. In this set of experiments we manipulate the regularization coefficient and look at the modularity under the earlier defined edge distance weighting strategies. We create embeddings of the Facebook Politicians graph with the parameter settings used in Subsection 4.3 and we plotted on Figure 4 the mean modularity based on 10 experimental repetitions.

The main insights based on the experiments are summarized as follows. The clustering performance of *DeepWalk* is more sensitive to the regularization coefficient than that of *GEMSEC*. We observe that *GEMSEC* has high quality results with nearly every

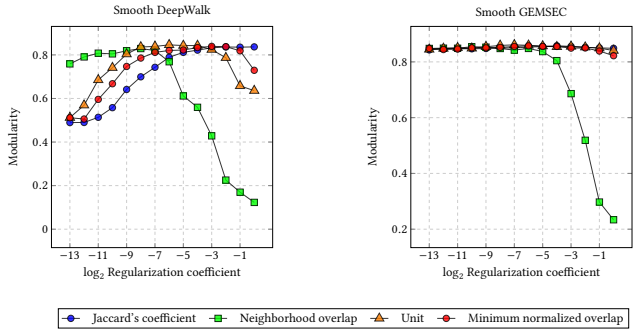


Figure 4: Sensitivity of cluster quality to smoothness regularization coefficient on the Facebook politicians network. *Smooth GEMSEC* shows a stronger robustness to regularization coefficient changes.

possible weighting strategy and setting of the regularization coefficient. Moreover, when the coefficient is too high the neighborhood overlap, minimum normalized overlap and unit weighting has considerable adverse effects on the clustering performance of *DeepWalk*. Finally, the clustering performance under the standard setting of the regularization coefficient seems to be optimal for both models.

4.5 Robustness to Perturbation

In a number of settings the graph that we observe contains noisy edges or edges among vertices are not present. In a social network people who do not know each other might share an edge or conversely users who do know someone decide not to be linked on purpose. As such applications are highly relevant for real-world applications we will test the robustness of *GEMSEC* to these types of perturbations and compare it to the *DeepWalk* variants.

Utilizing the Facebook Politicians graph we create synthetic graphs where a given fraction of nodes is randomly removed or added while the number of connected components is unchanged. Using these synthetic graphs we created a clustering of nodes and evaluated the modularity on the original graph. We used the hyperparameter settings from Subsection 4.3 and each experiment was repeated 10 times. The plots on Figure 5 report mean modularity and error bars correspond to two standard deviations.

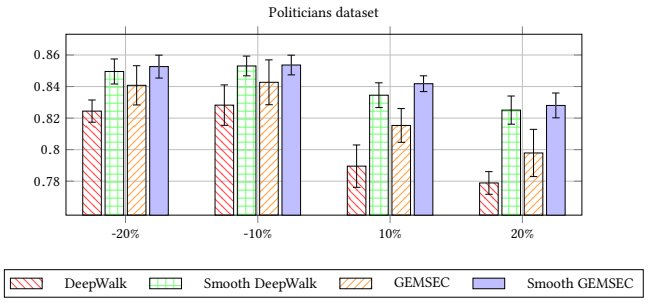


Figure 5: Sensitivity of cluster quality to randomized edge removal and addition measured by modularity. Smoothness regularization increases robustness to edge addition.

First, we can observe that all of the models are quite robust to edge removal — we see little difference between the modularity values when we increased the fraction of removed edges from 10% to 20%. On the contrary the addition of edges that are non-existent in the original graph has a considerable adverse effect on clustering performance. Second, we see that the use of smoothness regularization with Jaccard's index weighting mitigates this effect. This is due to the fact that noisy edges between randomly selected nodes are expected to have a zero neighborhood overlap. Because of this, for noise edges we do not have a constraint on making representations to be similar. Finally, it should be considered that even when we add a considerable amount of edges randomly we are able to perform comparably to the baselines.

4.6 Scalability

As scalability is an important aspect in real world applications we also investigated the scalability of our procedure. The quantity of interest was the time needed to do a single training epoch as the sampling procedure and the feature extraction is equivalent to that of *DeepWalk* (assuming that the regularization weights are precalculated). Our experiments were done on Erdos-Renyi graphs with a fixed average degree of 20. We created embeddings of this graph with both *DeepWalk* and *GEMSEC* variants and we used the baseline parameter settings to create embeddings. We calculated mean optimization runtime in seconds per epoch based on ten epochs. The logarithms of these averages as a function of the log node number are plotted on Figure 6.

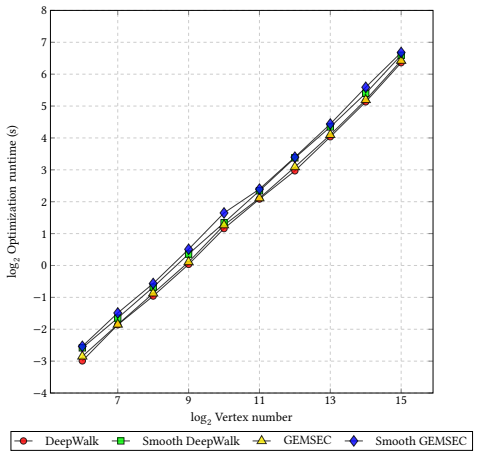


Figure 6: Sensitivity of optimization runtime to graph size measured by seconds. Our proposed models' optimization has linear runtime complexity in the number of nodes.

Most importantly, based on Figure 6 we can conclude that doubling the size of the graph doubles the time needed for optimizing *GEMSEC*. The linear nature of the proposed algorithm is well demonstrated by the figure. We also observe that learning the clustering and embedding jointly increases optimization time, which was expected. Besides this, the presence of smoothness regularization also inflates the time needed for optimization, but each of the smooth algorithms scales linearly with the number of vertices.

5 CONCLUSIONS

In this paper we proposed *GEMSEC* a novel algorithm that learns a graph embedding and a clustering of nodes jointly. The introduced model is a natural extension of earlier sequence based graph embedding procedures such as *DeepWalk* and *Node2vec* [11, 21]. We

reformulated the objective function used for sequence based graph embedding by adding a constraint that enforces nodes to be clustered around a fixed number of cluster centers in the embedding space. As an additional extension we introduced smoothness regularization of embeddings which can enforce the representation of nodes with highly overlapping neighbourhoods to be similar.

Empirical evaluation of the proposed graph clustering methods on Facebook data shows that the quality of extracted communities is quite high. Our methods outperform a number of strong baselines and also they give better results than simply clustering a *DeepWalk* embedding. We also demonstrate on social networks extracted from Deezer that *GEMSEC* with the right number of clusters is able to significantly improve the predictive accuracy on the downstream machine learning task of node labeling. Additional experiments establish that our proposed method is scalable, extracts high quality communities with a wide range of parameter settings and the clustering performance is robust to perturbation.

ACKNOWLEDGMENTS

Benedek Rozemberczki and Ryan Davies were supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1).

REFERENCES

[1] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 37–48.

[2] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. 2006. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 44–54.

[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations. In *AAAI*. 1145–1152.

[5] Sandro Cavallari, Vincent W Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2017. Learning community embedding with community detection and node embedding on graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 377–386.

[6] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding Community Structure in Very Large Networks. *Physical Review* 70, 6 (2004).

[7] Jimmy Ba Diederik P. Kingma. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR) (ICLR '15)*.

[8] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3 (2010), 75–174.

[9] Palash Goyal and Emilio Ferrara. 2017. Graph Embedding Techniques, Applications, and Performance: A Survey. *arXiv preprint arXiv:1705.02801* (2017).

[10] Steve Gregory. 2010. Finding Overlapping Communities in Networks by Label Propagation. *New Journal of Physics* 12, 10 (2010), 103018.

[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[12] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-Contrastive Estimation: A New Estimation Principle for Unnormalized Statistical Models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 297–304.

[13] David Hallac, Jure Leskovec, and Stephen Boyd. 2015. Network lasso: Clustering and optimization in large graphs. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 387–396.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[15] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907* (2016).

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*.

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Advances in neural information processing systems*. 3111–3119.

[18] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. 2016. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928* (2016).

[19] Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali, and Ploutarchos Spyridonos. 2012. Community detection in social media. *Data Mining and Knowledge Discovery* 24, 3 (2012), 515–554.

[20] Pons Pascal and Matthieu Latapy. 2005. *In International Symposium on Computer and Information Sciences*. Springer Berlin Heidelberg, Chapter Computing Communities in Large Networks Using Random Walks., 284–293.

[21] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online Learning of Social Representations.. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*.

[22] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. Don'T Walk, Skip!: Online Learning of Multi-scale Network Embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017 (ASONAM '17)*. ACM, New York, NY, USA, 258–265. https://doi.org/10.1145/3110025.3110086

[23] Bryan Perozzi and Steven Skiena. 2015. Exact Age Prediction in Social Networks. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15 Companion)*. ACM, New York, NY, USA, 91–92. https://doi.org/10.1145/2740908.2742765

[24] Tiago Pimentel, Adriano Veloso, and Nivio Ziviani. 2017. Unsupervised and Scalable Algorithm for Learning Node Representations. (2017).

[25] Doina Precup and Yee Whye Teh (Eds.). 2017. *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Proceedings of Machine Learning Research, Vol. 70. PMLR. http://jmlr.org/proceedings/papers/v70/

[26] Troy Raeder and Nitesh V Chawla. 2009. Modeling a store's product space as a social network. In *Social Network Analysis and Mining, 2009. ASONAM'09. International Conference on Advances in*. IEEE, 164–169.

[27] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. 2017. Struc2Vec: Learning Node Representations from Structural Identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 385–394. https://doi.org/10.1145/3097983.3098061

[28] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[29] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning Deep Representations for Graph Clustering.. In *AAAI*. 1293–1299.

[30] Twan Van Laarhoven and Elena Marchiori. 2012. Robust community detection methods with resolution parameter for complex detection in protein protein interaction networks. *Pattern Recognition in Bioinformatics* (2012), 1–13.

[31] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[32] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. 2012. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*. Springer, 639–655.

[33] Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Unsupervised Deep Embedding for Clustering Analysis. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 478–487. http://dl.acm.org/citation.cfm?id=3045390.3045442

[34] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *International Conference on Machine Learning*. 40–48.

[35] Shuhan Yuan, Xintao Wu, and Yang Xiang. 2017. SNE: Signed Network Embedding. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 183–195.