# Outlier Resistant Unsupervised Deep Architectures for Attributed Network Embedding

Sambaran Bandyopadhyay*
IBM Research
sambband@in.ibm.com

Lokesh N†
Indian Institute of Science, Bangalore
lokeshn@iisc.ac.in

Saley Vishal Vivek†
Indian Institute of Science, Bangalore
vishalsaley@iisc.ac.in

M. N. Murty
Indian Institute of Science, Bangalore
mnm@iisc.ac.in

## ABSTRACT

Attributed network embedding is the task to learn a lower dimensional vector representation of the nodes of an attributed network, which can be used further for downstream network mining tasks. Nodes in a network exhibit community structure and most of the network embedding algorithms work well when the nodes, along with their attributes, adhere to the community structure of the network. But real life networks come with community outlier nodes, which deviate significantly in terms of their link structure or attribute similarities from the other nodes of the community they belong to. These outlier nodes, if not processed carefully, can even affect the embeddings of the other nodes in the network. Thus, a node embedding framework for dealing with both the link structure and attributes in the presence of outliers in an unsupervised setting is practically important. In this work, we propose a deep unsupervised autoencoders based solution which minimizes the effect of outlier nodes while generating the network embedding. We use both stochastic gradient descent and closed form updates for faster optimization of the network parameters. We further explore the role of adversarial learning for this task, and propose a second unsupervised deep model which learns by discriminating the structure and the attribute based embeddings of the network and minimizes the effect of outliers in a coupled way. Our experiments show the merit of these deep models to detect outliers and also the superiority of the generated network embeddings for different downstream mining tasks. To the best of our knowledge, these are the first unsupervised non linear approaches that reduce the effect of the outlier nodes while generating Network Embedding.

## KEYWORDS

network representation learning, community outliers, adversarial learning, deep autoencoder, graph mining, social networks

**ACM Reference Format:**
Sambaran Bandyopadhyay, Lokesh N, Saley Vishal Vivek, and M. N. Murty. 2020. Outlier Resistant Unsupervised Deep Architectures for Attributed Network Embedding. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371788

## 1 INTRODUCTION

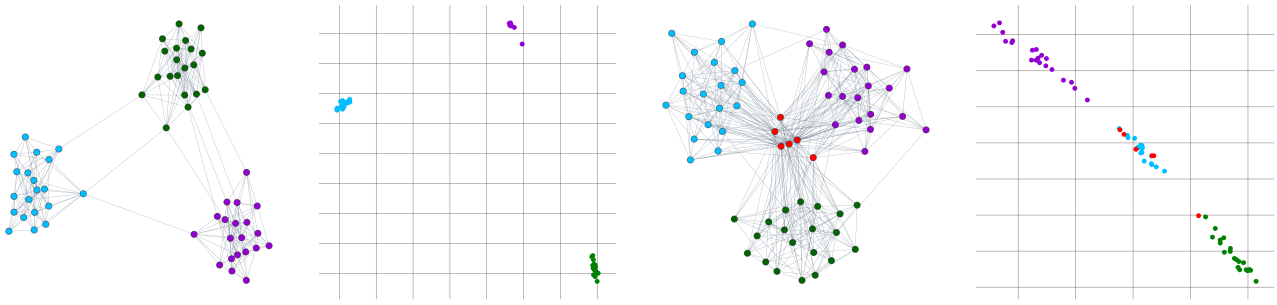
Network representation learning or network embedding [8, 23] is the task of mapping the nodes of a network (or graph) to a lower dimensional vector space. As different properties of the network are captured in the resulting space, downstream machine learning tasks perform better when these are used as feature vectors. An attributed network comes with additional attribute information present in each node. Multiple algorithms have been proposed in the literature to embed attributed networks [12, 32]. These embedding algorithms typically exploit the fact that attribute values of the nodes are highly correlated with the link structure of the network [20] and hence provide complementary information for network representation.

Nodes in an information network exhibit community structure [16]. Existing network embedding algorithms perform well when the nodes of the networks are well-connected in their respective communities and attributes are coherent with the link structure. But real-life networks come with nodes which violate the property of the community they belong to. Such a node can have edges with the nodes randomly from different communities or their attributes are similar to attributes of the nodes from other communities. These nodes are called community outliers [9]. We use the words outlier and the community outlier interchangeably in this work. For example, in social networks there are users who are randomly connected to other users. In a citation network, there can be research papers which cite other papers with highly varying levels of similarity. In these cases, homophily property [20] does not necessarily hold between the connected nodes. Moreover, a community outlier node might be a member of a community structurally, but its attribute values can be very different from the other members of the community and vice-versa. Real world attributed networks do have unlabelled outliers, as pointed out in [5, 17].

*S.B. is also affiliated to Indian Institute of Science, Bangalore.
†L.N and S.V.V. contributed equally.


*WSDM '20, February 3–7, 2020, Houston, TX, USA*

ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/3336191.3371788



(a) Synthetic Network (b) N2V embeddings of (a) (c) With Seeded Outliers (in red) (d) N2V embeddings of (c)

**Figure 1: Adverse effect of community outliers on the embedding of the regular nodes: (a) We use a small synthetic network with 3 communities and a total of 60 nodes (see Sec. 5.2). (b) We run node2vec on this synthetic graph with embedding dimension = 2 and plot the embeddings. It performs well by separating the communities far apart in the plot. (c) Then we insert only 6 community outlier nodes (in red) in the same network. These outliers have edges to all the three communities randomly. So they do not satisfy the community structure of the synthetic network. (d) Again we show embedding visualization of node2vec algorithm on the seeded synthetic dataset. Clearly, the seeded outliers pull the embeddings from different communities together, as compared to Sub-figure (b). As the original dataset is very well-structured, nodes can still be classified in the seeded version. But the distortion shows the adverse effect of the outliers on the embeddings of the regular nodes. Interestingly, (d) also shows that outliers get mixed up with the regular nodes, and thus mere post processing cannot detect them.**

Most of the existing network embedding algorithms do not handle these community outlier nodes explicitly while generating the node embeddings. As a result of that, outliers can heavily affect the embedding of the regular nodes in the network. This can be seen in Fig 1, where only few manually seeded outliers in a synthetic network distort the communities in the node2vec embedding space by pulling them together. This deterioration happens because the outlier nodes drive the random walks across communities and hence homophily property is violated in the resulting embeddings. Adverse effect of outliers on real world datasets are experimentally demonstrated in Section 5.6 as well. To overcome this problem, the effect of outlier nodes in the overall embedding objective needs to be minimized while generating the embeddings of the nodes. Thus, it is important to propose an integrated solution for node embedding and outlier detection for (attributed) network.

Recently, researchers have considered outliers while generating network embedding. A semi-supervised algorithm is proposed to detect outliers while generating the network embedding in [18]. But often it is difficult and expensive to get labelled data for the nodes of a network. [2] proposes an unsupervised algorithm which reduces the effect of outliers in network embedding. But this approach has two limitations. First, real world complex networks exhibit highly nonlinear behavior, which is difficult to capture using matrix factorization. Second, matrix factorization techniques do not scale for larger networks. To address these research gaps, we propose two unsupervised deep models. We call them as DONE (**D**eep **O**utlier aware attributed **N**etwork **E**mbedding) and AdONE (**Ad**versarial **ONE**) respectively. Following are the contributions we make:

- We propose an autoencoder based deep architecture (DONE) to minimize the effect of outliers for network embedding, in an unsupervised way. We use SGD, along with the derived closed form update rules for faster optimization of the parameters of the network. To the best of our knowledge, this is the first unsupervised deep architecture for outlier aware attributed network embedding.
- We propose another unsupervised deep architecture (AdONE) by exploiting the idea of adversarial learning for outlier aware network embedding. To the best of our knowledge, this is the first work to use adversarial training for this task.
- To show the superiority of the proposed algorithms, we conduct thorough experiments on both original and seeded versions of four publicly available datasets for three downstream network mining tasks. The source code of the proposed algorithms and additional materials are available at https://bit.ly/35A2xHs.

## 2 RELATED WORK

A comprehensive survey on network embedding can be found in [12, 30]. For the sake of completeness, we only cite some important literature related to our work. DeepWalk [23], LINE [25] and node2vec [8] are some of the first few popular node embedding algorithms, which preserve network node proximities in the embedding space by direct optimization or via random walks inspired by the skip-gram models [21] proposed in the natural language processing literature. Different matrix factorization based techniques for node embedding in attributed networks are proposed in [1, 13, 32]. The concept of generative adversarial learning has also been used in the context of node embedding in [29]. Recently, deep learning techniques gain much popularity for network embedding. A deep autoencoder for graph embedding is proposed in [3]. SDNE [28] proposes another autoencoder based approach to preserve different types of proximities in a network. The idea of using convolutional

neural networks (GCN) for graph embedding by repeatedly aggregating attribute values from neighbors has been proposed in [15, 22]. An extension of GCN with different types of node information aggregation methods (GraphSage) is developed in [11]. Another deep learning architecture for attributed network embedding is proposed in [6]. Attention mechanism for graph embedding is introduced in [26]. A GCN based approach by maximizing mutual information between patch representations and high-level summaries of a graph is proposed in DGI [27].

The above approaches do not process the outlier nodes explicitly and hence are prone to be affected by them. Recently, a semi-supervised approach SEANO [18] and an unsupervised matrix factorization based approach ONE [2] have been proposed for outlier detection and network embedding for attributed networks. But these approaches have their own limitations as discussed in Section 1. There are also recent works on detecting outliers (or anomalies) in attributed networks by analyzing the residual or reconstruction loss of the network via latent space [17] or deep models [5]. But they do not minimize the effect of outliers in the node embeddings, thus cannot prevent the deterioration of the embedding quality because of the outliers. We precisely address these research gaps in this paper, and propose scalable deep unsupervised solutions.

## 3 PROBLEM STATEMENT

An attributed information network is typically represented by a graph as $\mathcal{G} = (V, E, C)$, where $V = \{v_1, v_2, \cdots, v_N\}$ is the set of nodes (a.k.a. vertices). $E \subset \{(v_i, v_j)|v_i, v_j \in V\}$ is the set of edges between the vertexes. First order neighborhood of a node $i$ is denoted as $\mathcal{N}(v_i) = \{v_j \in V|(v_i, v_j) \in E\}$. The network can be directed or undirected, and weighted or unweighted. Let, $A$ is the $N \times N$ dimensional adjacency matrix of the graph $\mathcal{G}$. For a large network, the matrix $A$ is highly sparse in nature. Let $C$ be a $N \times D$ matrix with $\mathbf{c}_i$ as rows, where $\mathbf{c}_i \in \mathbb{R}^D$ is the attribute vector associated with node $v_i \in V$. $C_{id}$ is the value of the attribute $d$ for node $v_i$. For example, if there is only textual content in each node, $\mathbf{c}_i$ can be the tf-idf vector for the content of the node $v_i$.

Next, we discuss the set of outliers that we aim to detect in this paper. The anomalous behavior of the nodes can be captured by their link structure, attributes or by the combination of both. More formally, there are three types of community outliers [2] in an attributed network as shown in Figure 2: (a) **Structural Outlier**: The node is structurally connected to nodes from different communities, i.e., its structural neighborhood is inconsistent. (b) **Attribute Outlier**: The attributes of the node are similar to that of nodes from different communities, i.e., its attribute neighborhood is inconsistent. (c) **Combined Outlier**: Node belongs to a community structurally but it belongs to a different community in terms of attribute similarity. For a given network $\mathcal{G}$, our goal is to learn a node embedding function $f : v_i \mapsto \mathbf{h}_i \in \mathbb{R}^K$, that maps every vertex to a $K$ dimensional vector, where $K < min(N, D)$. The representations should preserve the underlying semantics of the network. Hence the nodes which are close to each other in topographical distance or similarity in attributes should have similar representations. As mentioned in Sec. 1, we also need to reduce the effect of outliers and aim to detect them in the process of network embedding, so that the embeddings of the other nodes in the network are robust.

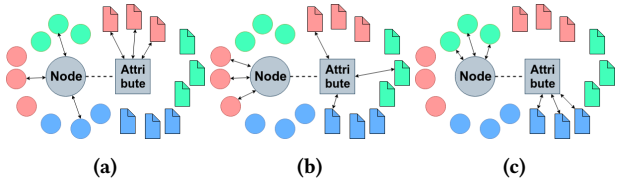



**Figure 2: Different types of outliers [2] present in an attributed network: (a) Structural Outlier, (b) Attribute Outlier and (c) Combined Outlier. Nodes (or attributes) having same color belong to the same community in terms of structure or attribute similarity.**

## 4 SOLUTION APPROACHES: DEEP MODELS

This section discusses the proposed deep architectures to solve the problem of outlier aware network representation. We do the following preprocessing of the input network first.

### 4.1 Network Preprocessing

Real life networks are highly sparse and come with missing connections between nodes. The rows of an adjacency matrix can only capture the observed links as they are. Motivated by the concept of page rank, we use random walk with restart [3, 24] to obtain a richer context and consequently preserve the higher order proximities in our proposed solution. Given the adjacency matrix $A$ of the network $\mathcal{G}$ (Sec. 3), the transition matrix can be obtained as $D^{-1}A$, where $D$ is a diagonal matrix with $D_{ii} = \sum_{j=1}^{N} a_{ij}$. Suppose, $P^t \in \mathbb{R}^{N \times N}$ represents the probability matrix where $(P_i^t)_j$ represent the probability of going to node $j$ after $t$ steps by a random walk starting from the node $i$, $t = 0, 1, \cdots, T$. $T$ is the maximum length of the truncated random walk. Clearly, $P_i^0$ ($i^{th}$ row of the matrix $P^0$) has $(P_i^0)_i = 1$ and all the other elements as 0. So, $P_i^t = rP_i^{t-1}[D^{-1}A] + (1-r)P_i^0$. Where $0 \leq r \leq 1$ with $(1-r)$ being the restart probability of the random walk from the starting node at any step. We take the average of all the matrices $P^1, \cdots P^T$ to capture the higher order proximities between the nodes. For the experiments, we set $r = 0.3$ and $T = 3$. Thus we use the rows of the following matrix $X \in \mathbb{R}^{N \times N}$ as the input to our proposed architectures: $X = \frac{1}{T} \sum_{t=1}^{T} P^t$.

### 4.2 Solution Approach: DONE

This is the first solution approach where we use two parallel autoencoders for link structure and attributes of the nodes respectively. As shown in Figure 3, let us refer the first autoencoder corresponding to the structure of the network as $Enc^s$ and that corresponding to the attributes as $Enc^a$. Henceforward, we always use superscript $s$ for structure and superscript $a$ for the attributes for all the functions if not mentioned otherwise. The input to the first autoencoder is $\mathbf{x}_i$ ($i^{th}$ row of the matrix $X$ in Sec. 4.1) and that to the second autoencoder is $c_i$ ($i^{th}$ row of the attribute matrix $C$ in Sec 3) for node $i$. There are $L$ layers in each of the encoders and decoders. We have used Leaky ReLU nonlinearity function with negative input slope $\alpha = 0.2$ in both the autoencoders. The embeddings of node $i$ with respect to the structure and attributes are $\mathbf{h}_i^s$ and $\mathbf{h}_i^a$, obtained from

the hidden (code) layers of $Enc^s$ and $Enc^a$ respectively, $\mathbf{h}_i^s, \mathbf{h}_i^a \in \mathbb{R}^K$. The reconstructed outputs of the autoencoders for node $i$ are $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{c}}_i$ respectively. Let us also call the set of all the parameters (all $\mathbf{W}$'s and $\mathbf{b}$'s) of the autoencoders as $\Theta$.

We also introduce outlier scores ($\in \mathbb{R}$) for each node corresponding to the three types of outliers as shown in Fig. 2. We denote them as $o_i^s$, $o_i^a$ and $o_i^{com}$ corresponding to structural, attribute and combined outliers respectively for node $i$, $i = 1, \cdots, N$. The set of all the outlier scores is denoted by $O$. For the sake of interpretability of outlierness, we assume:

$$\sum_{i=1}^{N} o_i^s = 1, \quad \sum_{i=1}^{N} o_i^a = 1, \quad \sum_{i=1}^{N} o_i^{com} = 1, \quad o_i^s, o_i^a, o_i^{com} > 0 \tag{1}$$

It is important to note that, if the outlier scores are not upper bounded, then they will all be assigned to $+\infty$ by the optimization in Eq. 7 or in Eq. 13 (as discussed later). Intuitively, we assume that total outlier score for each type of outlier is constant in the network. For a *perfect* network where there is no outlier present (for e.g., a graph with modular communities and no inter-community edges, where attributes are consistent and perfectly coherent with the link structure), outlier scores of all the nodes are equal to each other, $o_i^s = o_i^a = o_i^{com} = \frac{1}{N}, \forall i$. Outlier scores of each type also form a discrete probability distribution. For example, $o_i^s$ denotes the probability of the node $v_i$ to be a structural outlier.

Let us formulate the loss functions for this approach. Like most of the embedding algorithms, we also want to preserve different orders of proximities in the network. As the input to the structural autoencoder captures the local neighborhood of a node (Section 4.1), by minimizing this reconstruction loss $\sum_{i=1}^{N} ||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2^2$, one can preserve the higher order proximity in the network. But the presence of outliers can adversely affect the learning of the parameters of the network. So it is important to minimize their contribution in the learning process. Hence we reformulate the proximity loss as:

$$\mathcal{L}_{str}^{Prox} = \frac{1}{N} \sum_{i=1}^{N} log(\frac{1}{o_i^s}) ||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2^2 \tag{2}$$

Clearly larger the outlier score $o_i^s$ for some node $i$, smaller would be the value of $log(\frac{1}{o_i^s})$, and so the contribution to loss from this node would be less. The next component of the loss function is used to preserve homophily [20] in networks. Nodes which are connected by edges tend to have similar behavior and they should be close in the embedding space as well. Again, as structural outliers have connections to nodes from multiple communities randomly, their contribution to homophily loss should be less.

$$\mathcal{L}_{str}^{Hom} = \frac{1}{N} \sum_{i=1}^{N} log(\frac{1}{o_i^s}) \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} ||\mathbf{h}_i^s - \mathbf{h}_j^s||_2^2 \tag{3}$$

We divide the total loss over the neighbors by the degree of the node $v_i$ so that a node does not contribute significantly more because of its degree. With a similar motivation, for the attribute autoencoder, the following two losses can be formulated.

$$\mathcal{L}_{attr}^{Prox} = \frac{1}{N} \sum_{i=1}^{N} log(\frac{1}{o_i^a}) ||\mathbf{c}_i - \hat{\mathbf{c}}_i||_2^2 \tag{4}$$

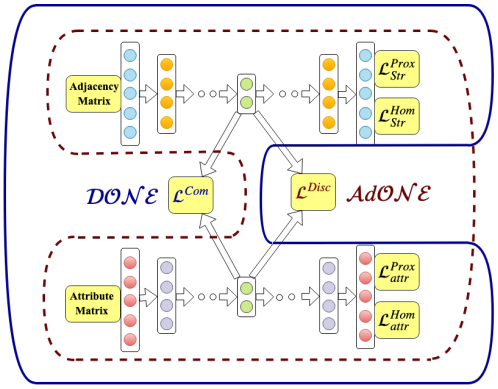



**Figure 3: DONE and AdONE Architectures: The architecture within the blue boundary (2 autoencoders, connected via $\mathcal{L}^{Com}$) is DONE. The architecture within the dotted red boundary (2 autoencoders, connected via $\mathcal{L}^{Disc}$) is AdONE**

$$\mathcal{L}_{attr}^{Hom} = \frac{1}{N} \sum_{i=1}^{N} log(\frac{1}{o_i^a}) \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} ||\mathbf{h}_i^a - \mathbf{h}_j^a||_2^2 \tag{5}$$

Please note that, we have used attribute outlier score $o_i^a$ for each node while formulating the loss for the attribute autoencoder. Next, from the homophily property [20], the link structure and node attributes of a node in a network are highly correlated. Hence it is important to use one as complimenting the other. Though we are getting embeddings corresponding to structure and attributes in the network, it is important to regularize them for each node. Hence we formulate the last component (combining structure and attributes) of the loss function as:

$$\mathcal{L}^{Com} = \frac{1}{N} \sum_{i=1}^{N} log(\frac{1}{o_i^{com}}) ||\mathbf{h}_i^s - \mathbf{h}_i^a||_2^2 \tag{6}$$

Again as explained in Figure 2, combined outliers are different in the link structure and attribute behaviors. So we minimize their contribution in the loss. So the total loss that we want to minimize with respect to the constraints on outlier scores in Eq. 1 is ($\alpha$'s being the positive weight factors):

$$\min_{\Theta, O} \mathcal{L}_{DONE} = \alpha_1 \mathcal{L}_{str}^{Prox} + \alpha_2 \mathcal{L}_{str}^{Hom} + \alpha_3 \mathcal{L}_{attr}^{Prox} + \alpha_4 \mathcal{L}_{attr}^{Hom} + \alpha_5 \mathcal{L}^{Com} \tag{7}$$

*4.2.1 Optimization and Training for DONE.* The set of parameters $\Theta$ of the autoencoders and the outlier scores $O$ of the nodes need to be learnt by minimizing the loss in Eq. 7 with the constrains in Eq. 1. We use ADAM (with default hyper-parameter setting) optimization technique [14] to learn the parameters of the autoencoders. Calculation of the exact homophily losses (in Eq. 3 and 5) are expensive, specially for the nodes with high degrees. So for each iterative update, we randomly sub-sample 2 nodes from the neighborhood of each node and take their average to approximate the complete average over the neighborhood. This also performs good experimentally. Unfortunately, the same gradient based optimization to learn the outlier scores turns out to be extremely slow. Hence we derive closed form update rules for them as follows.

It can be seen that the loss $\mathcal{L}_{DONE}$ is convex in each of the outlier scores of a node when all other variables are fixed. Hence we use alternating minimization technique to update each variable. We derive the update rule for the set of $o_i^s, \forall i$ first. The Lagrangian of Eq. 7 with respect to the constraint $\sum_{i=1}^{N} o_i^s = 1$ can be written as (ignoring the terms which do not involve $o_i^s$):

$$\mathbb{L} = \lambda\Big(\sum_{i=1}^{N} o_i^s - 1\Big) + \alpha_1\Big(\frac{1}{N}\sum_{i=1}^{N} log(\frac{1}{o_i^s})||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2^2\Big) + \alpha_2\Big(\frac{1}{N}\sum_{i=1}^{N} log(\frac{1}{o_i^s})\frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} ||\mathbf{h}_i^s - \mathbf{h}_j^s||_2^2\Big)$$

$\lambda \in \mathbb{R}$ is the Lagrangian constant. Equating the partial derivative of the above w.r.t. $o_i^s$ to 0, we obtain:

$$o_i^s = \frac{\alpha_1||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2^2 + \alpha_2 \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} ||\mathbf{h}_i^s - \mathbf{h}_j^s||_2^2}{N\lambda}$$

Using the fact, $\sum_{i=1}^{N} o_i^s = 1$, we get:

$$o_i^s = \frac{\alpha_1||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2^2 + \alpha_2 \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} ||\mathbf{h}_i^s - \mathbf{h}_j^s||_2^2}{\sum_{i=1}^{N}\Big(\alpha_1||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2^2 + \alpha_2 \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} ||\mathbf{h}_i^s - \mathbf{h}_j^s||_2^2\Big)} \quad (8)$$

Interestingly, the update rule for $o_i^s$ turns out to be very intuitive. At each iteration, it is proportional to the weighted sum of the reconstruction loss of $x_i$ (refer Eq. 2) and the average structural homophily loss over the neighborhood of the $i$th node. Similarly, for attribute and combined outliers:

$$o_i^a = \frac{\alpha_3||\mathbf{c}_i - \hat{\mathbf{c}}_i||_2^2 + \alpha_4 \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} ||\mathbf{h}_i^a - \mathbf{h}_j^a||_2^2}{\sum_{i=1}^{N}\Big(\alpha_3||\mathbf{c}_i - \hat{\mathbf{c}}_i||_2^2 + \alpha_4 \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} ||\mathbf{h}_i^a - \mathbf{h}_j^a||_2^2\Big)} \quad (9)$$

$$o_i^{com} = \frac{||\mathbf{h}_i^s - \mathbf{h}_i^a||_2^2}{\sum_{i=1}^{N} ||\mathbf{h}_i^s - \mathbf{h}_i^a||_2^2} \quad (10)$$

In eq. 8 and 9, the respective second terms in both numerator and denominator involve sum over the neighborhood. Similar to the calculation of homophily loss, we again use neighborhood sub-sampling to approximate the complete average over the neighborhood. Please note that, each denominator of the eq. 8, 9 and 10 involves a summation over all the nodes. But this sum is exactly same for all the nodes, and hence needs to be computed only once for a full iteration. For training, we first pretrain the autoencoders without the outlier scores. Then we alternately update the outliers scores in their respective closed form rules, and then update the parameters of the autoencoders using ADAM till the convergence. Final embedding of a node $i$ is obtained by concatenating the embeddings for structure and attributes as $h_i = h_i^s||h_i^a$.

**Time Complexity**: Assuming, the number of layers in the autoencoders as constant, updating the parameters of the autoencoders take $O(NK)$ time, due to the neighborhood sub-sampling. For updating the outlier scores in closed form solution, computation of the denominator for each type of outliers needs $O(NK)$ time (again due to the sub-sampling strategy) and computation of each outlier score for a node takes $O(K)$ time. Hence total time to update all the outlier scores take $O(NK)$ time. Thus each iteration of DONE takes $O(NK)$ time. It also converges fast on all the real life datasets we use in Section 5.

### 4.3 Solution Approach: AdONE

In this section, we propose an adversarial learning [7] based solution for outlier resistant network embedding. Adversarial training has recently been used for different machine learning applications such as active learning based sequence generation and labeling [4]. It has also been applied for network embedding on general graphs (without outliers) [29]. In an attributed network, it is important to align the embeddings corresponding to the link structure and node attributes so that they can complement each other. In DONE, we use weighted L2 norm (Eq. 6) to regularize the embedding from the structure and the attributes. But sometimes, a direct minimization of L2 norm can be too restrictive because L2 regularization brings the structure and attribute embeddings of a node to be very close with respect to the Euclidean Distance metric. This is not appreciable especially when the link structure and the attributes are not completely coherent with each other for a significant number of nodes in the network. Hence, we propose AdONE which uses a more flexible approach to address the alignment problem.

The key idea behind AdONE is the use of a discriminator for aligning the embeddings got from the structure and the attributes from the respective autoencoders. As shown in Figure 3, AdONE also employs two parallel autoencoders with the same configurations as that in DONE (Section 4.2). The embedding layers of both the autoencoders are connected with the discriminator. The task of the discriminator is to discriminate the embeddings coming from the first autoencoder $Enc^s$ (corresponding to the link structure) and that coming from the second autoencoder $Enc^a$ (corresponding to the attribute space). When the two autoencoders are pre-trained independently, the discriminator would be able to classify them easily after some training. Next, the extra task of the autoencoders would be updating themselves to fool the discriminator, so that it cannot distinguish the embeddings coming from two different sources. Thus there is a min-max game between the two encoders and the discriminator. The equilibrium of this game is attained when the Discriminator outputs equal probability for the structure and attribute embedding classes. At the end of this game, the structural and attribute embeddings $h_i^s$ and $h_i^a$ may still not be very close to each other in the Euclidean space. But, they would be aligned in the sense that the structure and attribute embeddings are very close to the decision boundary of the discriminator at equilibrium.

More formally, we use a two-layer neural network (dimension of the hidden layer being 16) as the discriminator. Let us denote the set of parameters for the discriminator as $\Theta_D$. We sample from the embedding space of the first autoencoder ($h_i^s \sim E^s$) and send them as the positive example for the discriminator, and corresponding sample of that from the second autoencoder ($h_i^a \sim E^a$) as the negative example. Following are the cost functions we propose to learn the parameters of discriminator and the autoencoders. First

one is the discriminator function that it aims to **maximize**:

$$\mathcal{L}^{Disc}(\Theta_D) = \frac{1}{N}\sum_{i=1}^{N}\Big(log(D(\mathbf{h}_i^s)) + log(1 - D(\mathbf{h}_i^a))\Big) \quad (11)$$

Clearly, the discriminator wants the probability output close to 1 for the samples from the first autoencoder, and close to 0 for the second one. But, the encoders $E^s$ and $E^a$ would try to fool the discriminator by **minimizing** the same function (along with Eq. 2 to 5), but weighted with the outlier scores to reduce their effects, defined as the alignment loss below:

$$\mathcal{L}^{Alg} = \frac{1}{N}\sum_{i=1}^{N} log(\frac{1}{o_i^{com}})\Big(log(D(\mathbf{h}_i^s)) + log(1 - D(\mathbf{h}_i^a))\Big) \quad (12)$$

Decreasing the contributions of the combined outliers is important. They have highly inconsistent link structure and attribute similarity as compared to the rest of the network. Hence, discriminator would be easily able to classify them and the alignment loss for the combined outliers would be very high. The gradient flow from these combined outliers (to the parameters of the autoencoders), if not controlled, could potentially affect not only the embeddings of the outliers but also the embedings of other regular nodes in the network. In AdONE, we minimize the effect of combined outliers using the term $\log(\frac{1}{o_i^{com}})$ in the alignment loss $\mathcal{L}^{Alg}$, Eq.12. The Structural and Attribute outliers would be managed by the terms $\frac{1}{o_i^s}$ and $\frac{1}{o_i^a}$ terms in the structure and attribute autoencoder respectively, as in DONE. Hence the total loss for AdONE ($\beta$'s being the weight factors) w.r.t. the constraints in Eq. 1 is :

$$\min_{\Theta,O} \ \mathcal{L}_{AdONE} \quad (13)$$
$$= \beta_1\mathcal{L}_{str}^{Prox} + \beta_2\mathcal{L}_{str}^{Hom} + \beta_3\mathcal{L}_{attr}^{Prox} + \beta_4\mathcal{L}_{attr}^{Hom} + \beta_5\mathcal{L}^{Alg}$$

Similar to DONE, we obtain the final embeddings in AdONE for a node $i$ by concatenating the embeddings for structure and attributes as $h_i = h_i^s||h_i^a$. Importance of the adversarial training for AdONE is experimentally shown in Section 5.7.

*4.3.1 Optimization and Training for AdONE.* Again we use ADAM with default hyper parameter setting to update the set of parameters of the discriminator neural network (Eq. 11), and that of the autoencoders. Similar to the case of DONE, we derive closed form update rule for updating the outlier scores to make the process computationally efficient. We skip the details as the derivation is similar to that of DONE. For training AdONE, first we pretrain the autoencoders independently. Next for the adversarial learning, we run multiple updates of the discriminator to improve itself by maximizing $\mathcal{L}^{Alg}$. Then we update each autoencoder once based on minimizing the total loss in Eq. 13. We repeat this process till the discriminator outputs almost equal probability for all the sample embeddings. Time complexity of one full iteration AdONE is same as DONE, which is $O(NK)$.

# 5 EXPERIMENTAL EVALUATION

We conduct detailed experimentation in this section.

**Table 1: Summary of the datasets used in this paper. Unseeded refers to the publicly available datasets. Seeded refers to the publicly available datasets with manually planted community outliers as discussed in Section. 5.1**

| Dataset | #Nodes | | #Edges | | #Labels | #Attributes |
|---|---|---|---|---|---|---|
| | Unseeded | Seeded | Unseeded | Seeded | | |
| WebKB | 877 | 919 | 1434 | 1662 | 5 | 1703 |
| Cora | 2708 | 2843 | 5429 | 6269 | 7 | 1433 |
| Citeseer | 3312 | 3477 | 4598 | 5319 | 6 | 3703 |
| Pubmed | 19717 | 20701 | 44325 | 49523 | 3 | 500 |

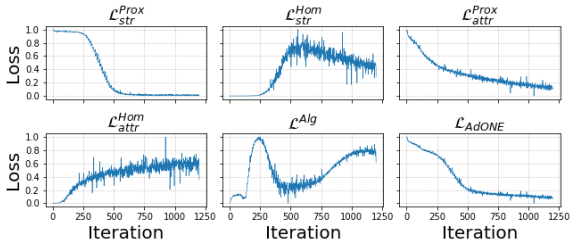


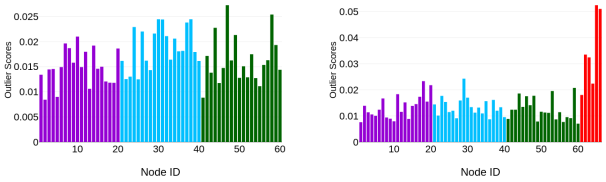

**Figure 4: Change of (normalized between 0 to 1) loss components of AdONE over iterations on Cora Dataset**

## 5.1 Datasets Used and Seeding Outliers

There is no publicly available attributed network dataset with ground truth outliers. So we manually planted a total of 5% outliers (with equal numbers for each type as shown in Figure 2) in publicly available attributed network datasets (https://linqs.soe.ucsc.edu/data). We follow the strategy used in [2] to ensure that seeded outliers are close to real outliers. The seeding process involves: (1) computing the probability distribution of number of nodes in each class, (2) selecting a class using these probabilities. For a structural outlier: (3) plant an outlier node in the selected class such that the node has ($m$ + /−10%) of edges connecting nodes from the remaining (unselected) classes where $m$ is the average degree of a node in the selected class and (4) the attributes of the structural outlier node are made semantically consistent with the keywords sampled from the selected class. A similar approach is employed for seeding the attribute outliers (sampling attributes randomly from different classes) and combined outliers (sampling edges and attributes from two different classes respectively). These outlier nodes have characteristics similar to the normal nodes in terms of degree, number of nonzero attributes, so that they cannot be detected trivially. The statistics of the datasets are given in Table 1.

## 5.2 Baseline Algorithms and Setup

We use node2vec [8], Line [25], SDNE [28], GraphSage [11] (unsupervised version), DGI [27], SEANO [18] (with 20% of labelled data), ONE [2] and Dominant [5] as the baseline algorithms to be compared with. Dominant is a very recent GCN based autoencoder approach to detect outliers from the attributed networks, but it does not minimize the effect of outliers on the node embeddings. We use the default parameter settings in the publicly available implementations of the respective baseline algorithms. To make a fair




**(a) Without Outliers** **(b) With Outliers**

**Figure 5: It shows the average of the three outlier scores, generated by DONE, of the nodes in the synthetic network in Fig. 1. In particular, (a) represents the outlier scores for 60 nodes in Fig. 1(a) and (b) represents the outlier scores for 66 nodes in Fig. 1(b). Clearly, the variation of outlier scores is high in (b) as the outlier scores for the manually inserted outliers in Fig. 1(b) are significantly more than the regular nodes.**

comparison, we have not included baseline algorithms which are semi-supervised in nature (except for SEANO which is explicitly designed for outlier detection). We also consider only those baselines for which the source code is available in public.

We set the embedding dimension ($K$) to be 32 for all algorithms. For both DONE and AdONE, the encoders and the decoders both have 2 layers for the first three datasets and 3 layers for the Pubmed dataset as it is relatively larger. We found that giving equal importance to all the components of DONE and AdONE leads to smooth convergence and good performance. So we always set the values of all the hyper-parameters in Eq. 7 and 13 to 1.Fig. 4 shows the faster convergence of loss for AdONE and its different components. The same can be observed for DONE as well. This is because we adopt a mixed update strategy by using ADAM for the neural network parameters and closed form update rules for the outliers. We train the models once and fix the embeddings for all the downstream tasks for DONE and AdONE. All downstream algorithms are run 5 times for each experiment and the average performance is reported.

To give more insights, we run DONE on the synthetic network shown in Fig. 1. The unseeded network is generated using Stochastic Block Model approach [10] and we manually insert 6 community outliers into the seeded version. We run DONE on both the unseeded and seeded versions (we pass the adjacency matrix as input for the attribute encoder also). From the plots of the outlier scores ($o_i = \frac{o_i^s + o_i^a + o_i^{com}}{3}$) in Fig. 5, it is clear that our approach is able to learn the outliers successfully in both the cases.

## 5.3 Outlier Detection

Outlier detection in attributed network is extremely important. Both DONE and AdONE produce outlier scores along with the node embeddings. We take a weighted average of the three outlier scores to generate a ranked list L of the nodes. Larger the score, the larger is the outlierness of a node. Among the baselines, only ONE, SEANO and Dominant produce direct outlier scores. Following the strategy of [18], for other baselines, we use isolation forest algorithm [19] on the generated embeddings to get outlier scores of the nodes. We only consider seeded datasets for this experiment as unseeded versions do not have the labelled ground truth outliers. Each seeded dataset has 5% outliers. So we plot the outlier recall from the top 5% to 25% of the nodes in the ranked list (L) with respect to the seeded outliers. Figure 6 shows that DONE and AdONE are the highest performers on WebKB and Pubmed, whereas Dominant performs best for Cora and Citeseer in detecting outliers. Dominant, though performs well for outlier detection as it employs a powerful GCN encoder, fails on the embedding based tasks (in Section 5.4 and 5.5) because it does not reduce the effect of outliers on the node embeddings. Most of the standard graph embedding algorithms like node2vec and DGI suffer as they do not process outliers while generating the embeddings. AdONE is able to outperform DONE in most of the cases because link structure is often not fully coherent with the node attributes of the networks. So the direct minimization of L2 norm suffers more compared to the adversarial learning.

## 5.4 Community Detection

Community detection or node clustering is another popular task in information network analysis. We give the node embeddings as input to KMeans++ algorithm to cluster the nodes. To judge the quality of clustering, we use unsupervised clustering accuracy [2, 31]. For the unseeded datasets in Fig.7(a), Node2Vec, SEANO, DONE, AdONE are the good performers. For Seeded datasets in Fig.7(b), AdONE turns out to be the best performing algorithm (except for the Pubmed where ONE outperforms AdONE marginally). Even though some algorithms outperform DONE and AdONE in some unseeded datasets, presence of just 5% outliers completely affect the embedding quality. This is primarily because the injected outliers hinder the community structure in the dataset. Our approaches on the other hand are outlier resistant and hence the outliers don't influence the embeddings of the other good nodes substantially.

## 5.5 Node Classification

The next task we consider here is node classification. We vary the training size from 10% to 50%. We train a logistic regression classifier on the training set of embeddings (along with the class labels) and check the performance on the test set by using Micro F1 scores. Figure 8 shows the performance of node classification on the seeded datasets. Again AdONE and DONE perform the best on all the datasets (AdONE is better on Pubmed). Among the baselines, SEANO performs better on Cora and Citeseer while ONE performs better on WebKB and Pubmed. As expected, other baselines mostly fail for this task as well because of the effect of outliers.

## 5.6 Adverse Effect of Community Outliers

This section provides results on the real world datasets to show the adverse effect of outliers on the embedding of the regular nodes via downstream mining tasks. Here we compare the performance of an algorithm on unseeded vs. the corresponding seeded dataset. Figure 7 already shows that the presence of community outliers deteriorates the clustering accuracy. Table 2 further shows the adverse effect on the node classification results. For example, node2vec is one of the best performing algorithm on the unseeded Pubmed for node-classification. But the presence of only 5% outliers deteriorates its performance by more than 46% on the seeded Pubmed.

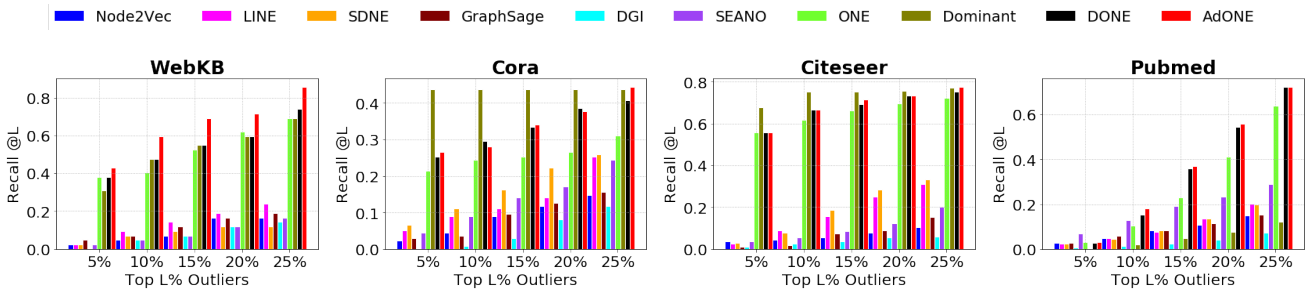
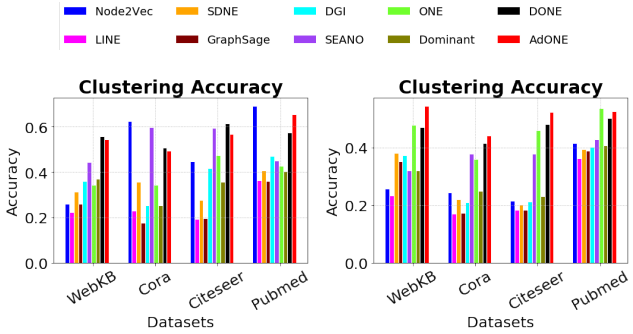



**Figure 6: Recall at top L% from the ranked list of outliers by different Embedding algorithms.**




**(a) Without Outliers** **(b) With Outliers**

**Figure 7: Unsupervised clustering accuracy of KMeans++ on the node embeddings generated by different algorithms**

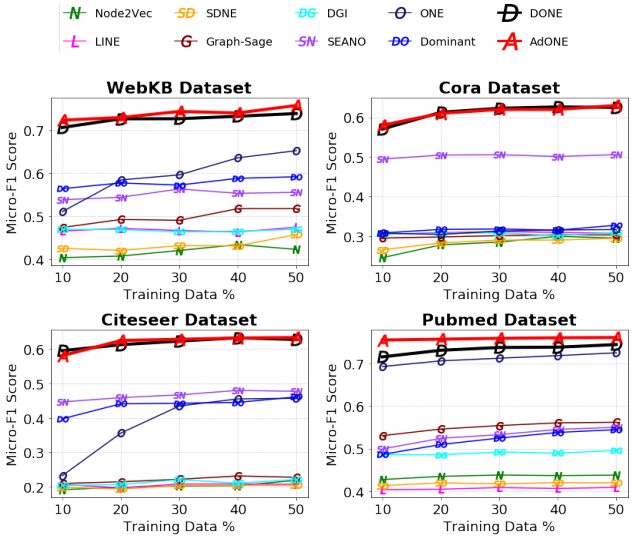



**Figure 8: Accuracy of node classification with logistic regression on the seeded datasets**

## 5.7 Experimental Insight for AdONE

As we have seen, in AdONE, to fool the discriminator both structure and attribute embeddings of a node should be close to the decision boundary of the discriminator (to align them). To understand the merit of adversarial learning in AdONE, we conduct a small experiment. On Cora dataset, we learn the node embeddings using AdONE with the discriminator **disabled** by equating $\beta_5 = 0$ in AdONE objective in Eq. 13. Then, with the obtained embeddings we perform classification task and compare the results with that of the regular AdONE (i.e. with the discriminator). We show the micro-F1 classification score for the both the cases, along with the performance gain due to the use of the discriminator in Table 3. In particular, the gain is more significant when the training size is less. So it is evident that having a discriminator to do the adversarial training helps improve the embedding quality for AdONE.

**Table 2: Adverse effect of community outliers by comparing the Micro-F1 scores for node classification with 50% training size for different embedding algorithms on *seeded* (S) and *unseeded* (Un-S) datasets. Overall, DONE and AdONE perform the best and the performance deterioration due to outliers is also less (in the higher performance zone) for them.**

| Algorithm \ Dataset | WebKB | | Cora | | Citeseer | | Pubmed | |
|---|---|---|---|---|---|---|---|---|
| | Un-S | S | Un-S | S | Un-S | S | Un-S | S |
| Node2Vec | 0.47 | 0.42 | 0.63 | 0.29 | 0.54 | 0.21 | **0.80** | 0.43 |
| LINE | 0.48 | 0.47 | 0.32 | 0.30 | 0.24 | 0.20 | 0.53 | 0.40 |
| SDNE | 0.51 | 0.45 | 0.57 | 0.29 | 0.39 | 0.20 | 0.63 | 0.41 |
| GraphSage | 0.48 | 0.51 | 0.32 | 0.30 | 0.24 | 0.22 | 0.42 | 0.56 |
| DGI | 0.49 | 0.46 | 0.54 | 0.30 | 0.67 | 0.22 | 0.79 | 0.49 |
| SEANO | 0.48 | 0.55 | 0.59 | 0.50 | 0.64 | 0.47 | 0.60 | 0.55 |
| ONE | 0.72 | 0.65 | 0.43 | 0.31 | 0.53 | 0.45 | 0.75 | 0.72 |
| Dominant | 0.63 | 0.59 | 0.38 | 0.32 | 0.71 | 0.46 | 0.78 | 0.54 |
| DONE | **0.80** | 0.73 | **0.74** | 0.62 | **0.71** | 0.62 | 0.79 | 0.74 |
| AdONE | 0.77 | **0.75** | 0.67 | **0.63** | 0.68 | **0.63** | **0.80** | **0.76** |

**Table 3: Micro-F1 scores for node classification on Cora (with outlier seeded) for different training sizes with and without adversarial learning for AdONE.**

| Method | Training Size % | | | | |
|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 |
| AdONE without Discriminator | 0.49 | 0.55 | 0.58 | 0.59 | 0.61 |
| AdONE with Discriminator | **0.58** | **0.61** | **0.62** | **0.62** | **0.63** |
| % Performance Gain | 18.37 | 10.9 | 6.9 | 5.08 | 3.28 |

### 5.8 Parameter Sensitivity Analysis

We check the sensitivity of the proposed algorithms with respect to embedding dimension $K$. We run DONE and AdONE on Cora dataset for $K = \{8, 16, 32, 64, 128\}$ and test the performance for node classification task. Because we concatenate the structure and attribute embeddings ($h_i = h_i^s || h_i^a$), embedding layer has $\frac{K}{2}$ dimensions in each autoencoder. The results are reported in the Figure 9. We can observe that $K = 32$ is good enough for Cora Dataset for both DONE and AdONE.

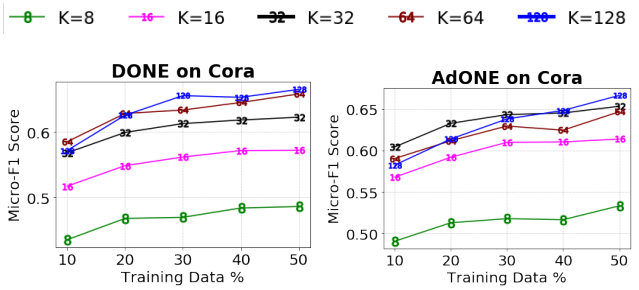



**Figure 9: Parameter Sensitivity Analysis of the embedding dimension $K$ on Cora (seeded) dataset for DONE and AdONE.**

## 6 CONCLUSION

In this work, we propose two unsupervised deep neural network based architectures for detecting and minimizing the effect of community outliers while generating node embeddings in attributed networks. Outliers can be there in all the real world networks and experimentally we have shown their adverse effect on all the standard embedding algorithms. Our algorithms are computationally faster and scalable and can deal with larger datasets. In future, we would like to extend these algorithms to dynamic and multiplex networks which are popular in many real world applications.

## REFERENCES


[1] Sambaran Bandyopadhyay, Harsh Kara, Aswin Kannan, and M Narasimha Murty. 2018. FSCNMF: Fusing Structure and Content via Non-negative Matrix Factorization for Embedding Information Networks. *arXiv preprint arXiv:1804.05313* (2018).

[2] Sambaran Bandyopadhyay, N Lokesh, and M Narasimha Murty. 2019. Outlier Aware Network Embedding for Attributed Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 12–19.

[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[4] Yue Deng, KaWai Chen, Yilin Shen, and Hongxia Jin. 2018. Adversarial Active Learning for Sequences Labeling and Generation.. In *IJCAI*. 4012–4018.

[5] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. 2019. Deep Anomaly Detection on Attributed Networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 594–602.

[6] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding.. In *IJCAI*. 3364–3370.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[8] 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*. ACM, 855–864.

[9] Manish Gupta, Jing Gao, Yizhou Sun, and Jiawei Han. 2012. Integrating community matching and outlier detection for mining evolutionary community outliers. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 859–867.

[10] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. Exploring Network Structure, Dynamics, and Function Using NetworkX. *Proceedings of the 7th Python in Science Conference*.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1025–1035.

[12] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *arXiv preprint arXiv:1709.05584* (2017).

[13] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 633–641.

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[16] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[17] Jundong Li, Harsh Dani, Xia Hu, and Huan Liu. 2017. Radar: Residual Analysis for Anomaly Detection in Attributed Networks.. In *IJCAI*. 2152–2158.

[18] Jiongqian Liang, Peter Jacobs, Jiankai Sun, and Srinivasan Parthasarathy. 2018. Semi-supervised embedding in attributed networks with outliers. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 153–161.

[19] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 413–422.

[20] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* 27, 1 (2001), 415–444.

[21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[22] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*. 2014–2023.

[23] 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*. ACM, 701–710.

[24] Daniel A Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the STOC*, Vol. 4.

[25] 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.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJXMpikCZ

[27] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep graph infomax. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rklz9iAcKQ

[28] 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.

[29] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. GraphGAN: Graph Representation Learning With Generative Adversarial Nets. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16611

[30] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

[31] Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*. 478–487.

[32] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network Representation Learning with Rich Text Information.. In *IJCAI*. 2111–2117.