Unsupervised Attributed Network Embedding via Cross Fusion

Guosheng Pan[1], Yuan Yao[1], Hanghang Tong[2], Feng Xu[1], Jian Lu[1]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, China
[2]University of Illinois at Urbana-Champaign, USA
pgs@smail.nju.edu.cn,{y.yao,xf,lj}@nju.edu.cn,htong@illinois.edu

ABSTRACT

Attributed network embedding aims to learn low dimensional node representations by combining both the network's topological structure and node attributes. Most of the existing methods either propagate the attributes over the network structure or learn the node representations by an encoder-decoder framework. However, propagation based methods tend to prefer network structure to node attributes, whereas encoder-decoder methods tend to ignore the longer connections beyond the immediate neighbors. In order to address these limitations while enjoying the best of the two worlds, we design cross fusion layers for unsupervised attributed network embedding. Specifically, we first construct two separate views to handle network structure and node attributes, and then design cross fusion layers to allow flexible information exchange and integration between the two views. The key design goals of the cross fusion layers are three-fold: 1) allowing critical information to be propagated along the network structure, 2) encoding the heterogeneity in the local neighborhood of each node during propagation, and 3) incorporating an additional node attribute channel so that the attribute information will not be overshadowed by the structure view. Extensive experiments on three datasets and three downstream tasks demonstrate the effectiveness of the proposed method.

CCS CONCEPTS

• **Information systems** → **Data mining**; *Social networks.*

KEYWORDS

Network embedding, network structure, local community, node attributes

ACM Reference Format:
Guosheng Pan[1], Yuan Yao[1], Hanghang Tong[2], Feng Xu[1], Jian Lu[1]. 2021. Unsupervised Attributed Network Embedding via Cross Fusion. In *Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM '21), March 8–12, 2021, Virtual Event, Israel.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WSDM '21, March 8–12, 2021, Virtual Event, Israel
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8297-7/21/03…$15.00
https://doi.org/10.1145/XXXXXX.XXXXXX

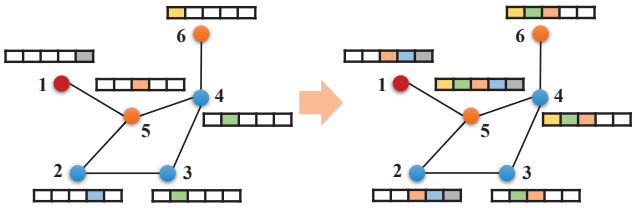


Figure 1: An illustrative example. The color of boxes represents the existence of node attributes, and the node color indicates class label. The left is the original attributed network and the right is the version after applying the propagation based methods. Nodes 5 and 6 with the same label/color can be easily predicted due to the propagated attributes, while Nodes 1 and 2 with different labels/colors become more indistinguishable after propagation.

1 INTRODUCTION

Attributed network embedding has attracted growing attention in recent years. The basic idea is to learn low dimensional representations/embeddings for network nodes by combining both the network's topological structure and the attributes of nodes. The learned embeddings have been shown to be effective in many downstream tasks including node classification [46], node clustering [20], and link prediction [40].

To date, existing attributed network embedding methods can be roughly divided into three categories. The first category is based on the *matrix factorization* framework (e.g., TADW [46] and AANE [15]) where matrices containing the network structure and node attributes are constructed and jointly factorized. However, these shallow models might fall short in capturing the non-linearity of the two information sources [12]. The second category is related to *graph neural networks* (e.g., GCN [18] and GraphSAGE [14]). These methods, which are referred to as *propagation based* methods in this work, recursively aggregate node representations by applying the graph convolution operation among each node's immediate neighbors. While such smoothing techniques have achieved state-of-the-art results on several graph analysis tasks, they tend to suffer from the "over-smoothing" problem [21, 49], i.e., they may lose discriminative features in the original node attributes and hence prefer network structure to node attributes. Methods in the third category adopt the *encoder-decoder* framework (e.g., DANE [12] and ANRL [52]). Compared to the second category, node attributes are specially handled so that their discriminative features can be maintained; however, methods in this category treat network structure in a fixed manner and thus cannot propagate critical information along the network edges, beyond the immediate neighbors (we name this limitation as the "under-smoothing" problem).

To illustrate the advantages and disadvantages of the existing methods, we present an illustrative example in Fig. 1. In this example, we propagate the node attributes along the network edges as the propagation based methods do. Node 5 and Node 6 (with the same label as indicated by color in the figure) show the advantage of such propagation based methods. That is, although they have quite different attributes and network structure before propagation, these two nodes share three identical attributes after propagation, making it easier to predict their labels. However, propagation could also bring side effects. For example, Node 1 and Node 2 (with different labels) have their respective discriminative attributes (as indicated by grey and light blue boxes) before propagation. After propagation, these two nodes tend to have the same attributes, rendering a more difficult prediction task to distinguish between them. In this case, an independent treatment on the node attributes as methods in the third category would be more appropriate.

In this paper, we envision that propagation based methods and encoder-decoder methods are inherently complementary, and thus ask: *how can we design an attributed network embedding model that enjoys the best of the two worlds?* To answer this question, we propose an attributed network embedding model with two separate views to handle network structure and node attributes. Based on the two views, we design the cross fusion layers to allow flexible information exchange and integration between them. To be specific, there are three key advantages of our cross fusion design. First, the cross fusion layer can be naturally stacked to mimic the graph convolution operation so that critical information can be propagated along the network edges (i.e., mitigating the "under-smoothing" problem of encoder-decoder methods). Second, when aggregating the node representations from immediate neighbors, we pay special attention to the local community structure to enable the assignment of different weights to different neighbors. Third, we use an additional view to encode node attributes so that the attribute information will not be overshadowed by the structure view (i.e., mitigating the "over-smoothing" problem of propagation based methods). With the learned embeddings from different views, instead of simply concatenating them like existing work, we add a view weighting layer to allow each node learn their own relative weights from the two views. Finally, we learn the node representations in an unsupervised way by considering the reconstructions of both network structure and node attributes.

To show the effectiveness of the proposed method, we conduct experimental evaluations by utilizing the learned embeddings for three downstream tasks (i.e., node classification, node clustering, and visualization) on three widely-used benchmark datasets. The results show that the proposed method significantly outperforms the existing methods in terms of prediction accuracy. Additionally, the proposed unsupervised method can even achieve comparable results with semi-supervised ones in node classification. Finally, consistent with the above analysis, we have experimentally found that propagation based methods perform better on datasets where network structure matters more, and encoder-decoder methods become better when node attributes play a more prominent role.

In summary, the main contributions of this paper include:

- A multi-view framework CFane with cross fusion layers for attributed network embedding. CFane seamlessly enjoys the advantages of both propagation based methods and encoder-decoder methods.
- Extensive experiments demonstrating the superior performance of the proposed method. For example, the proposed CFane achieves up to 60.4% improvement over its best competitors in the node clustering task.

Table 1: Symbols.

Symbol	Definition and Description
G	the input network
$\mathcal{V}$	the node set of the input network
$\mathcal{E}$	the edge set of the input network
$\mathbf{A}$	the adjacency matrix of the input network
$\mathbf{X}$	the attributes of all nodes
$\mathbf{Y}$	the learned representations of all nodes
$\mathbf{x}_i$	the attributes of node v_i
$\mathbf{y}_i$	the learned representation of node v_i
$\mathbf{v}_1$	the node representation of structure view
$\mathbf{v}_2$	the node representation of attribute view
$\mathbf{c}, \mathbf{f}$	the intermediate representation vectors
k	the attribute dimension
d	the embedding dimension

The rest of the paper is organized as follows. Section 2 provides the problem statement. Section 3 describes the proposed method, and Section 4 presents the experimental results. Section 5 reviews related work, and Section 6 concludes.

2 PROBLEM STATEMENT

In this section, we state the problem definition. Table 1 lists the main symbols we use throughout the paper. Following conventions, we use calligraphic letters to represent sets. For example, the node set and edge set of a network are denoted as $\mathcal{V}$ and $\mathcal{E}$, respectively. We use bold capital letters to indicate matrices and bold lowercase letters to indicate (row) vectors. For example, we use $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ to denote the adjacency matrix of the input network, and $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|\times k}$ to denote the matrix of node attributes where k is the attribute dimension. We also use the corresponding lowercase to indicate the row vectors of the matrix. For example, we use $\mathbf{x}_i$ to denote the i-th row of $\mathbf{X}$. With the above notations, we first define an attributed network as follows.

Definition 1. Attributed Networks. An attributed network is denoted as $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V}$ is the node set and $\mathcal{E}$ is the edge set. $\mathbf{X}$ contains the node attributes, with its i-th row $\mathbf{x}_i$ describing the attributes for the i-th node $v_i \in \mathcal{V}$.

Attributed network embedding aims to learn the node representations by taking into account both the network structure and the node attributes. We use $\mathbf{Y} \in \mathbb{R}^{|\mathcal{V}|\times d}$ to denote all the learned node representations where d is the dimension of the learned embeddings. $\mathbf{y}_i \in \mathbb{R}^{1\times d}$ indicates the i-th row of $\mathbf{Y}$. We then define the attributed network embedding problem as follows.

Definition 2. Attributed Network Embedding. Given an attributed network $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, attributed network embedding aims to learn a function $f : v_i \mapsto \mathbf{y}_i$ that maps each node $v_i \in \mathcal{V}$ to a low-dimensional representation vector $\mathbf{y}_i$.

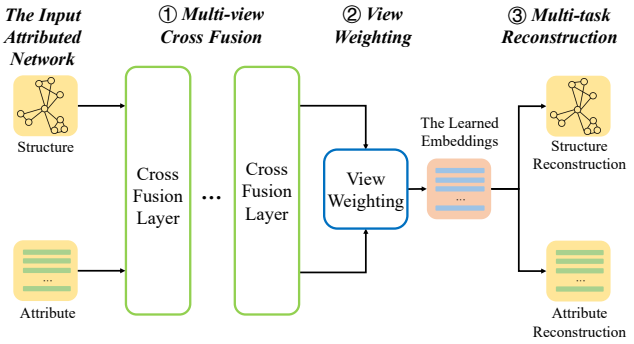


Figure 2: The overview of CFane.

3 THE PROPOSED APPROACH

In this section, we present the proposed approach CFane.

3.1 Model Overview

Fig. 2 shows the overview of the proposed CFane. In particular, to allow special treatment on node attributes, we first input the network structure and node attributes into two different views. Then the key issue becomes how to flexibly exchange and integrate information between the two views so as to enjoy the advantages from both propagation based methods and encoder-decoder methods. For this purpose, the core of CFane is the design of cross fusion layers with three key advantages as mentioned in introduction, i.e., 1) propagating critical information along network structure, 2) encoding neighbor heterogeneity during propagation and aggregation, and 3) maintaining discriminative features in node attributes. After the cross fusion layers, we further introduce a view weighting layer to let each node integrate its own representation with different weights from the two views. Finally, to obtain the learned embeddings in an unsupervised setting, we formulate multi-task reconstruction objective functions to guide the embeddings to preserve both network structure and node attributes.

3.2 Cross Fusion Layer

We next describe the proposed cross fusion layer as illustrated in Fig. 3. It takes both the network structure and node attributes as input. In the following, we take the n-th cross fusion layer as an example, and use subscripts to indicate the two views for clarity. For example, we use two vectors $\mathbf{v}_1^{(n)}, \mathbf{v}_2^{(n)} \in \mathbb{R}^{1\times d^{(n)}}$ to denote the representations of each node in the current layer for the network structure view and the node attribute view, respectively. We will remove the superscript for brevity when it is clear from the context.

(A) Ego-network partition and aggregation. On one hand, due to the state-of-the-art performance of propagation based methods, we aim to take advantage of these methods by following the idea of recursively aggregating node representations among each node's immediate neighbors. On the other hand, when aggregating the node representations from immediate neighbors, different neighbors may have different impact/weights on the aggregated node representations, making it necessary to distinguish their roles. Consequently, to encode such heterogeneity among immediate neighbors, a natural choice is to learn different weights for different neighbors. In the following, we introduce a flexible aggregation method based on the partition in the ego-network [9].

An ego-network is a subgraph induced by the immediate neighbors of each node, defined as follows,

$$G[u] = (\mathcal{N}_u, \mathcal{E} \cap \mathcal{N}_u \times \mathcal{N}_u) \tag{1}$$

where $\mathcal{N}_u$ is the set of immediate neighbors of node u, and $\times$ means the Cartesian product. We then partition the ego-network of each node into several communities as follows,

$$par(G[u]) = \{\mathcal{N}_u^1, \mathcal{N}_u^2, ..., \mathcal{N}_u^{t_u}\} \tag{2}$$

where par is the partition function, $G[u]$ is the ego-network of node u as defined in Eq. (1), $\mathcal{N}_u^i$ contains the set of nodes in the i-th partition, and t_u is the number of resulting partitions of node u's ego-network.

Based on the partition results, we adopt a hierarchical smoothing method to aggregate the representations of each node from its immediate neighbors. First, we assign the same weight for nodes in the same community. For example, given the j-th community of node u, we use the average aggregation as follows,

$$\mathbf{c}_{\mathcal{N}_u^j}^{(n)} = \frac{1}{|\mathcal{N}_u^j| + 1} \sum_{i \in \mathcal{N}_u^j \cup \{u\}} \mathbf{v}_{1,i}^{(n)} \tag{3}$$

where $\mathbf{v}_{1,i}^{(n)} \in \mathbb{R}^{1\times d^{(n)}}$ indicates the current representation of node i in the network structure view. Next, we combine t_u representation vectors by learning different weights to different communities in the ego-network. In particular, we compute the weighted average representations over the communities as follows,

$$\mathbf{c}_1^{(n)} = \sum_{j \in [1, t_u]} (\gamma_j \cdot \mathbf{c}_{\mathcal{N}_u^j}^{(n)}) \tag{4}$$

where $\mathbf{c}_1^{(n)} \in \mathbb{R}^{1\times d^{(n)}}$ stands for the intermediate node representations after the ego-network aggregation, and γ_j is the importance coefficient computed by the following attention mechanism,

$$\gamma_j = \frac{\exp((\mathbf{v}_{1,u}^{(n)} \parallel \mathbf{c}_{\mathcal{N}_u^j}^{(n)})\mathbf{b}_\gamma^T)}{\sum_{i \in [1, t_u]} \exp((\mathbf{v}_{1,u}^{(n)} \parallel \mathbf{c}_{\mathcal{N}_u^i}^{(n)})\mathbf{b}_\gamma^T)} \tag{5}$$

where row vector $\mathbf{b}_\gamma$ is the parameter and $\parallel$ means the concatenation operation.

For the partition function par, since the size of ego-networks is usually small, we follow existing work [10] by treating each connected component of the ego-network as a community in this work. This setting helps encode the local community structure into the final network embeddings. Note that we can also use other partition functions. For example, if we put all the neighbors into one community, the above aggregation method degenerates to the GCN [18] model; if we treat each neighbor as a partition, the above method resembles the GAT [37] model. In other words, the proposed aggregation method provides flexibility in terms of balancing between the complexity (i.e., parameter size) and efficiency.

(B) Feature exchange between views. In addition to encoding neighbor heterogeneity, the other two design goals of CFane are to propagate critical information along network structure and maintain discriminative features in node attributes. For these purposes, we keep an additional view to encode the node attributes, and utilize the self-attention mechanism [36] to exchange features between

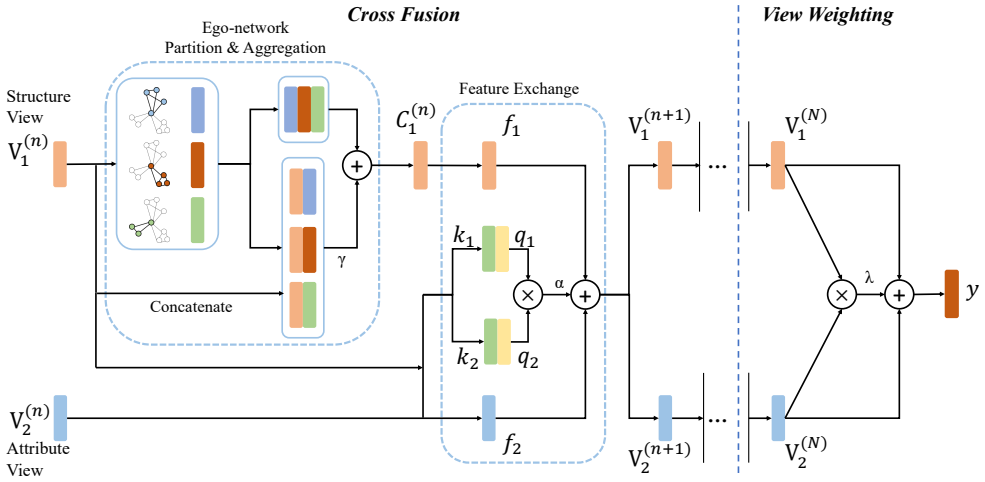


Figure 3: Cross Fusion and View Weighting.

views. The intuitions are two-fold. First, self-attention can learn the importance of the two views, without having one view overshadowed the other. Second, self-attention can output one vector for each view, so that the cross fusion layer can be naturally stacked.

Specifically, we first transform the features of the two views via a projection matrix $\mathbf{W}^F \in \mathbb{R}^{d^{(n)} \times d^{(n+1)}}$ as follows,

$$\mathbf{f}_1 = \mathbf{c}_1^{(n)} \mathbf{W}^F, \qquad \mathbf{f}_2 = \mathbf{v}_2^{(n)} \mathbf{W}^F \tag{6}$$

where $\mathbf{c}_1^{(n)}, \mathbf{v}_2^{(n)} \in \mathbb{R}^{1 \times d^{(n)}}$ indicate the two representations for each node in the current layer, $\mathbf{f}_1, \mathbf{f}_2 \in \mathbb{R}^{1 \times d^{(n+1)}}$ are the intermediate results, and $d^{(n+1)}$ is the embedding dimension of the next layer. Typically, $d^{(n+1)}$ is smaller than $d^{(n)}$ meaning that $\mathbf{W}^F$ helps to reduce the dimension size. Next, following standard self-attention, we calculate the query vector $\mathbf{q}$ and key vector $\mathbf{k}$ for each node,

$$\begin{aligned} \mathbf{q}_1 &= \mathbf{c}_1^{(n)} \mathbf{W}^Q, \quad \mathbf{q}_2 = \mathbf{v}_2^{(n)} \mathbf{W}^Q \\ \mathbf{k}_1 &= \mathbf{c}_1^{(n)} \mathbf{W}^K, \quad \mathbf{k}_2 = \mathbf{v}_2^{(n)} \mathbf{W}^K \end{aligned} \tag{7}$$

where $\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{d^{(n)} \times d^{(n+1)}}$ denote the transformation matrices for query vectors and key vectors, respectively. Then, we fuse the new node representations $\mathbf{v}_1^{(n+1)}$ and $\mathbf{v}_2^{(n+1)}$ with the following computation,

$$\begin{aligned} \alpha_{i,j} &= \frac{\exp(\mathbf{q}_i \mathbf{k}_j^T)}{\sum_{\mathbf{k} \in \{\mathbf{k}_1, \mathbf{k}_2\}} \exp(\mathbf{q}_i \mathbf{k}^T)} \\ \mathbf{v}_1^{(n+1)} &= \mu(\alpha_{1,1}\mathbf{f}_1 + \alpha_{1,2}\mathbf{f}_2) \\ \mathbf{v}_2^{(n+1)} &= \mu(\alpha_{2,1}\mathbf{f}_1 + \alpha_{2,2}\mathbf{f}_2) \end{aligned} \tag{8}$$

where $\alpha_{\cdot,\cdot}$ denotes the relative weights of the intermediate features $\mathbf{f}_1$ and $\mathbf{f}_2$ for the new node representations, and μ indicates the activation function. In this work, to add non-linear relationships between layers, we use the LeakyReLU as the default μ. In our experiments, we observed similar results of other non-linear activation functions such as ReLU.

3.3 View Weighting

After cross fusion layers, we combine the outputs of two views into a unified representation matrix $\mathbf{Y}$. Existing work mainly concatenates the representations from different views. However, the network structure view could be more informative for some nodes, while the node attributes may contribute more for others. In this work, we propose a view weighting layer (also shown in Fig. 3) to allow each node to choose their own relative weights between the two views. That is,

$$\mathbf{y} = \lambda_1 \mathbf{v}_1^{(N)} + \lambda_2 \mathbf{v}_2^{(N)} \tag{9}$$

where we suppose N cross fusion layers, and $\mathbf{v}_1^{(N)}$ and $\mathbf{v}_2^{(N)}$ are the node representations from the last layer. View weighting weights λ_1 and λ_2 are computed as follows,

$$\lambda_i = \frac{\exp(\mathbf{v}_i^{(N)} \mathbf{b}_\lambda^T)}{\exp(\mathbf{v}_1^{(N)} \mathbf{b}_\lambda^T) + \exp(\mathbf{v}_2^{(N)} \mathbf{b}_\lambda^T)} \tag{10}$$

where row vector $\mathbf{b}_\lambda$ is the parameter.

3.4 Training

Finally, to learn the parameters in our model as well as to make the learned embeddings more generic, we use multiple objective functions for preserving both the network structure and node attributes.

For the network structure, we aim to minimize the sum of the negative logarithmic probability of the node pairs in the same context,

$$\mathcal{L}_{skip-gram} = -\sum_{(i,j) \in C} \log \sigma(\mathbf{y}_j \mathbf{y}_i^T) \tag{11}$$

where $\mathbf{y}_i$ is the learned embedding for node v_i, and σ is the sigmoid function. We adopt the standard skip-gram model [27] to sample the node context C via truncated random walks and conduct negative sampling. For node attributes, we directly reconstruct them from the learned embeddings. We reconstruct the node attributes by

adding three feed-forward layers[1] with the following loss function,

$$\mathcal{L}_{recons} = \sum_{i=1}^{|\mathcal{V}|} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2 \tag{12}$$

where $\mathbf{x}_i$ and $\hat{\mathbf{x}}_i$ are the original attributes and reconstructed attributes of node v_i, respectively. Finally, we combine the two reconstruction tasks as the final objective function.

$$\mathcal{L} = \mathcal{L}_{skip-gram} + \beta \mathcal{L}_{recons} \tag{13}$$

where β is a balancing hyper-parameter.

3.5 Discussions and Analyisis

Comparison with existing multi-view encoder-decoder methods. Some existing work also adopts the multi-view framework for attributed network embedding (e.g., MVC-DNE [47] and DANE [12]). Compared to the proposed framework, the limitations of these two methods are two-fold. First, both MVC-DNE and DANE directly adopt auto-encoders which prevents propagating and smoothing features along the network structure; in contrast, we take advantage of the propagation based methods by smoothing the neighbors' features in the structure view. Second, they simply concatenate the learned embeddings of the two auto-encoders which ignores the relative weights/importance of different views; in contrast, our model is likely to assign higher weights to key characteristics in the original node attributes, which will be in turn propagated and preserved to the future layers.

Comparisons with existing propagation based methods. Similar to the graph convolution operator in existing propagation based GNNs [18, 37, 44], the information from immediate/one-hop neighbors is aggregated in one cross fusion layer. If we stack multiple layers, we can potentially propagate critical information to long-distance neighbors. Different from existing GNNs, we encode the local community structure in one-hop neighbors, and add an additional channel to encode node attributes to mitigate the over-smoothing problem (i.e., the attribute information will not be over-shadowed by the network structure).

Complexity analysis and computation speedup. By ignoring the layer number and the dimension size which are usually small constants, CFane shares the same time complexity $O(|\mathcal{V}| + |\mathcal{E}|)$ with GCN [18] and GAT [37], i.e., scaling linearly w.r.t. the number of edges and nodes in the network. We could also speed up the computation in several aspects. For example, in Eq. (7), we could use $\mathbf{v}_1^{(n)}$ instead of $\mathbf{c}_1^{(n)}$ for better efficiency. By doing so, we can compute self-attention in parallel with the ego-net partition and aggregation. For $\mathbf{f}_1$, we can compute it from right to left as the second dimension of $\mathbf{W}^F$ is usually much smaller than that of $\mathbf{V}_1$.

4 EXPERIMENTAL EVALUATIONS

In this section, we present the experimental results. We evaluate the effectiveness of the learned node representations in three tasks including node classification, clustering, and visualization.

Table 2: Statistics of the datasets.

Dataset	Nodes	Edges	Attributes	Classes
Cora	2708	5429	1433	7
Citeseer	3312	4732	3703	6
BlogCatalog	5196	171743	8189	6

4.1 Experimental Setup

Datasets. In our experiment, we use three widely-used attributed network datasets [15, 46]: *Cora*, *Citeseer*, and *BlogCatalog*. Cora and Citeseer are paper citation networks and BlogCatalog is an interaction network between bloggers. The statistics of these datasets are summarized in Table 2.

Comparison Methods. Since CFane is unsupervised, we first compare it with the following unsupervised attributed network embedding baselines.

- *DeepWalk* [30]. This is a classic network embedding method preserving the network structure only.
- *HARP* [4]. HARP extends DeepWalk by considering the hierarchical structure in networks.
- *AutoEncoder*. While the above two methods use the network structure only, we also compare the case when only node attributes are used via an auto-encoder.
- *TADW* [46]. TADW incorporates node features and network structure into a matrix factorization framework.
- *GAE and VGAE* [17]. These two methods are based on graph convolutional networks to propagate attributes along network edges. They can be seen as unsupervised GCNs [18].
- *GraphSAGE* [14]. It extends GCN by defining several aggregators to aggregate the attributes from a sampled set of neighbors. We use its unsupervised version with mean-aggregator by default.
- *DGI* [38]. DGI is an unsupervised representation learning method that adapts the mutual information maximization idea from image domain to graph domain.
- *ANRL* [52]. ANRL uses one encoder for node attributes and two decoders for node attributes and node neighborhood.
- *DANE* [12]. It employs two auto-encoders for network structure and node attributes, and then concatenates the two resulting embeddings.

To show the effectiveness of CFane, We also compare with some semi-supervised methods in the node classification task.

- *GCN* [14]. GCN is a semi-supervised network embedding method that applies average aggregation in the local neighborhood.
- *GAT* [37]. GAT learns the aggregation weights in the neighborhood by a node-level attention mechanism.

Parameters and Reproducibility. For all the baseline methods, we use the implementations provided by either their authors or open source libraries.[2] By default, we stack two cross fusion layers ($N = 2$) following GCN. The output dimensions of the two layers are 256 and 128, respectively. For fairness, the final embedding

[1] The size of the feed-forward layers depends on the attribute dimension of the input network.

[2] For the proposed method CFane, we will make the code publicly available upon acceptance.

Table 3: Node classification results. The proposed CFane significantly outperforms the existing methods in most cases. The bold numbers mean that the proposed CFane significantly outperforms all the competitors with p-values < 0.001.

Datasets		Cora			Citeseer			BlogCatalog		
Labeled Nodes		10%	30%	50%	10%	30%	50%	10%	30%	50%
Macro-F1	DeepWalk	0.729	0.771	0.785	0.442	0.505	0.513	0.496	0.548	0.573
	HARP	0.716	0.771	0.785	0.475	0.509	0.522	0.505	0.541	0.555
	AutoEncoder	0.587	0.665	0.693	0.594	0.633	0.649	0.707	0.733	0.741
	TADW	0.673	0.716	0.733	0.578	0.638	0.653	0.742	0.753	0.759
	GAE	0.779	0.807	0.815	0.595	0.624	0.634	-	-	-
	VGAE	0.770	0.816	0.826	0.533	0.594	0.614	-	-	-
	GraphSAGE	0.536	0.675	0.708	0.484	0.524	0.536	0.423	0.462	0.475
	DGI	0.311	0.604	0.674	0.579	0.627	0.631	0.621	0.735	0.747
	ANRL	0.719	0.748	0.759	0.660	0.682	0.688	0.694	0.705	0.709
	DANE	0.750	0.811	0.822	0.605	0.649	0.660	0.749	0.768	0.772
	CFane	**0.814**	**0.842**	**0.847**	0.669	**0.697**	**0.705**	**0.776**	**0.788**	**0.791**
Micro-F1	DeepWalk	0.740	0.795	0.808	0.482	0.555	0.575	0.585	0.646	0.676
	HARP	0.733	0.782	0.794	0.517	0.556	0.574	0.595	0.637	0.654
	AutoEncoder	0.627	0.694	0.717	0.644	0.680	0.696	0.826	0.857	0.866
	TADW	0.701	0.742	0.757	0.614	0.681	0.699	0.867	0.880	0.887
	GAE	0.790	0.818	0.825	0.652	0.687	0.699	-	-	-
	VGAE	0.785	0.827	0.836	0.579	0.644	0.662	-	-	-
	GraphSAGE	0.626	0.713	0.735	0.560	0.600	0.611	0.511	0.557	0.570
	DGI	0.484	0.709	0.758	0.678	0.726	0.732	0.748	0.861	0.874
	ANRL	0.747	0.769	0.777	0.717	0.732	0.739	0.815	0.828	0.832
	DANE	0.779	0.825	0.835	0.673	0.716	0.725	0.877	0.898	0.903
	CFane	**0.832**	**0.853**	**0.858**	**0.733**	**0.753**	**0.758**	**0.908**	**0.921**	**0.925**

dimension d of all methods is set to 128. For methods that concatenate two vectors as the final result, we set the dimension of each vector to 64. For the β parameter, we search it through {0.5, 1, 2} via cross-validation, and fix $\beta = 2$ on BlogCatalog, $\beta = 0.5$ on the other two datasets for simplicity. Other hyper-parameters are set as the default values provided in the corresponding papers. For training CFane, we need to learn parameters $\mathbf{W}^F$, $\mathbf{W}^Q$, $\mathbf{W}^K$ in each cross fusion layer, and parameters $\mathbf{b}_\gamma$ and $\mathbf{b}_\lambda$ for ego-network aggregation and view weighting. There are in total 4.3M, 10.1M, and 21.5M parameters for Cora, Citeseer, and BlogCatalog, respectively. All the experiments are run on a server with 8 cores of Intel Core CPU 4.00GHz and 64GB RAM. Note that we do not report the results of GAE and VGAE on BlogCatalog as they run out of the memory.

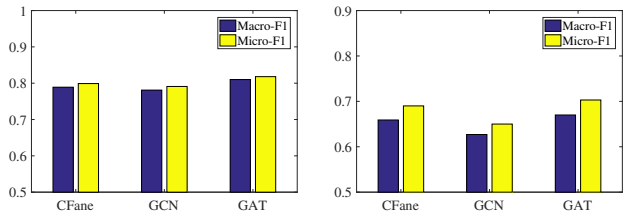
4.2 Experimental Results

(A) Effectiveness Comparisons in Node Classification. For node classification, we randomly select 10%, 30%, 50% labeled data as training set and use the remaining nodes as test set. Following existing work [30], we train a logistic regression classifier on the learned embeddings to predict the node labels. We repeat this process for 10 times and report the average Micro-F1 and Macro-F1 scores as the evaluation metrics. The results are shown in Table 3.

There are several observations from the table. First, the proposed CFane outperforms all the competitors in all cases. For example, on the Cora data, compared with the best competitor (i.e., DANE), CFane improves it by 2.8% to 8.5% in terms of Macro-F1/Micro-F1 scores; on BlogCatalog, the relative improvements range from 2.4% to 3.6% compared to DANE. We also conduct a significance test and the results show that most of the improvements are significant with p-values < 0.001. The reasons for the superior performance of CFane are as follows. CFane is better than the first three baselines (DeepWalk, HARP, and AutoEncoder) as they consider either network structure or node attributes only. For TADW, it integrates network structure and node attributes in a shallow matrix factorization framework, and thus cannot capture the non-linearity of the two information sources. CFane is better than GAE, VGAE, GraphSAGE, and DGI, as these propagation based methods tend to prefer network structure to node attributes (see more discussions in the following two paragraphs). For ANRL and DANE, their performance is inferior to CFane as they treat network structure in a fixed manner. In contrast, CFane can better handle network structure by propagating the attribute information to neighbors.

Second, DeepWalk and HARP use network structure only and AutoEncoder uses node attributes only. We can see that DeepWalk and HARP perform relatively well on Cora, whereas AutoEncoder is better on Citeseer and BlogCatalog. This result indicates that node attributes play a more important role on Citeseer and BlogCatalog, and network structure matters more on Cora.

Third, as discussed above, propagation based methods (e.g., GAE and VGAE) tend to put more emphasis on the network structure compared to node attributes. This is consistent with the observation that GAE and VGAE perform close to or even better than ANRL and DANE on Cora where the network structure is relatively more important, yet they perform significantly worse than ANRL and DANE on Citeseer where node attributes are more important.



(a) Cora (b) Citeseer

Figure 4: Comparison with semi-supervised methods. The unsupervised CFane achieves comparable results with the semi-supervised competitors.

Table 4: Clustering results. The proposed CFane significantly outperforms all the competitors in most cases with p-values < 0.001.

Datasets		Cora	Citeseer	BlogCatalog
NMI	Deepwalk	0.374	0.132	0.234
	HARP	0.367	0.131	0.187
	AutoEncoder	0.247	0.165	0.121
	TADW	0.375	0.286	0.066
	GAE	0.454	0.225	-
	VGAE	0.462	0.193	-
	GraphSAGE	0.340	0.171	0.111
	DGI	0.504	0.435	0.455
	ANRL	0.376	0.337	0.312
	DANE	0.483	0.188	0.414
	CFane	**0.577**	**0.447**	**0.730**
Purity	Deepwalk	0.583	0.401	0.423
	HARP	0.547	0.371	0.346
	AutoEncoder	0.475	0.394	0.325
	TADW	0.593	0.518	0.205
	GAE	0.618	0.405	-
	VGAE	0.609	0.361	-
	GraphSAGE	0.565	0.454	0.369
	DGI	0.694	0.713	0.595
	ANRL	0.575	0.575	0.455
	DANE	0.654	0.430	0.585
	CFane	**0.743**	0.701	**0.890**

(B) Comparison with Semi-supervised Competitors. We further compare our unsupervised CFane with semi-supervised methods GCN and GAT. In this experiment, we follow the experimental setting of GCN and GAT by randomly selecting 20 labels per class as known labels. The average Micro-F1 and Macro-F1 scores of 10 runs on Cora and Citeseer are shown in Table 4. As we can see from the table, CFane achieves comparable results with the semi-supervised competitors.

(C) Effectiveness Comparisons in Node Clustering. Next, we evaluate the performance of the learned embeddings in the clustering task. Specifically, we first run the K-means clustering algorithm on the embeddings generated by all the compared methods, and then evaluate the clustering results with normalized mutual information (NMI) and purity metrics. The results are shown in Table 4.

Table 5: The ablation study results. The four design choices of CFane are useful for the learned embeddings.

Variants	Classification		Clustering	
	Macro-F1	Micro-F1	NMI	Purity
CFane	0.791	0.925	0.730	0.890
CFane − CF	0.769	0.899	0.462	0.611
CFane − VW	0.767	0.897	0.587	0.756
CFane − NR	0.741	0.866	0.201	0.378
CFane − AR	0.774	0.905	0.666	0.856

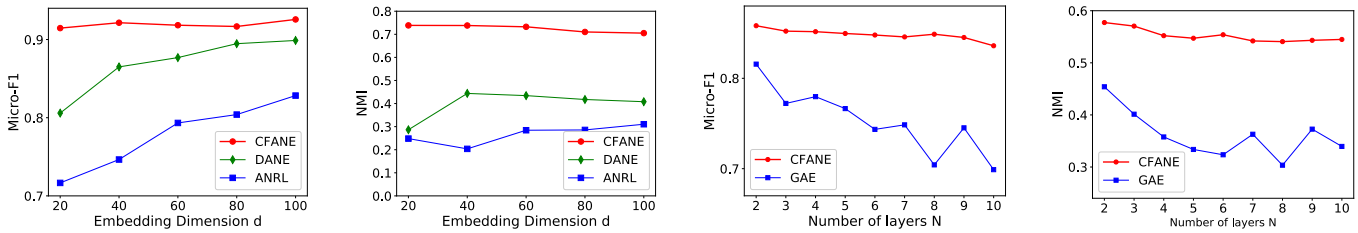
As we can see, the proposed CFane significantly outperforms all the competitors in most cases. For example, in terms of the NMI metric, CFane achieves 14.5%, 2.8%, and 60.4% relative improvements compared to the best competitor (DGI) in Cora, Citeseer, and BlogCatalog, respectively.

(D) Ablation Study. Here, we conduct an ablation study by deleting some of the design choices shown in Fig. 2. The classification and clustering results on BlogCatalog are shown in Table 5. For all the ablation studies and parameter sensitivity studies, similar results are observed in other datasets and are thus omitted for brevity. In the table, 'CFane − CF' means substituting the cross fusion layers in CFane with separate GCN and auto-encoder to deal with network structure and node attributes, 'CFane − VW' means deleting the view weighting layer, 'CFane − NR' means deleting the network reconstruction loss, and 'CFane − AR' means deleting the attribute reconstruction in the loss function. As we can see from the table, all the four design choices are effective in terms of improving the embedding results for both node classification and node clustering.

(E) Parameter Sensitivity. We also study the sensitivity of the two major hyper-parameters (i.e., embedding dimension d and cross fusion layer number N) in CFane. For d, we compare CFane with two best competitors DANE and ANRL, and the node classification and clustering results (we report Micro-F1 and NMI scores for brevity) with 50% training data on BlogCatalog are shown in Fig. 5(a) and Fig. 5(b), respectively. As we can see, our method consistently outperforms the two competitors as d varies.

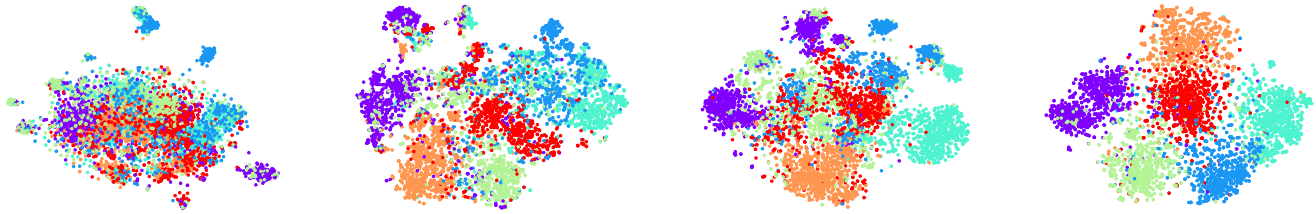
For layer number N, we compare CFane with propagation based method GAE. Since GAE runs out of memory on BlogCatalog, we report the results on Cora instead in Fig. 5(c) and Fig. 5(d). As we can see, when more layers are stacked, the accuracy of GAE decreases dramatically due to the over-smoothing problem, whereas CFane remains relatively stable. This result is consistent with our analysis that the cross fusion design helps to mitigate the over-smoothing problem by preserving some discriminative features in node attributes into future layers.

(F) Visualization Results. Finally, we visualize the learned node representations in a 2D space with t-SNE [26]. Fig. 6 shows the results on BlogCatalog. In the figures, we also present results from DeepWalk, ANRL, and DANE for comparison. As we can see, the result of DeepWalk is relatively cluttered as it does not consider node attributes. Both ANRL and DANE can identify several types of nodes in clusters yet many nodes of the same type are scattered in the space. CFane achieves the best visualization results, as most nodes with the same labels are closely gathered for all the labels.



(a) d in node classification (b) d in node clustering (c) N in node classification (d) N in node clustering

Figure 5: The parameter sensitivity results. The proposed CFane consistently outperforms the two best competitors as embedding dimension d varies, and CFane is more stable than GAE when we stack more than two layers.



(a) DeepWalk (b) ANRL (c) DANE (d) CFane

Figure 6: Visualization results of the learned node representations on BlogCatalog. CFane achieves better visualization results.

5 RELATED WORK

In this section, we briefly review the related work.

Plain Network Embedding. Plain network embedding aims to learn node representations by preserving only the structural properties of the network. For example, DeepWalk [30] employs random walks to generate node sequences and learns the embeddings through the skip-gram model; LINE [34] defines objective functions to preserve the first- and second-order similarities between nodes; node2vec [13] extends DeepWalk by introducing DFS and BFS based searching strategies; NetMF [31] unifies these existing methods into a matrix factorization framework. Several researchers also propose to incorporate the community structure [6, 25, 41] when such information is available. Other typical examples include [2, 4, 32, 39]. In contrast to these existing methods, our focus is on the embedding of attributed networks, which further considers node attributes.

Attributed Network Embedding. We divide existing attributed network embedding work into three categories. The first category is built upon the skip-gram model or equivalently the matrix factorization framework [19]. For example, TADW [46] uses inductive matrix completion to incorporate both network structure and the textual features on nodes; AANE [15] factorizes the attribute affinity matrix with regularization constraints from network edges. Other examples in this category include [1, 33, 48]. These methods usually adopt shallow models and thus cannot well capture the non-linearity of both network structure and node attributes [12].

The second category is based on graph neural networks [17, 18], where node attributes are propagated and smoothed along the network edges. For example, GraphSAGE [14] provides several aggregators to aggregate the information from sampled neighbors; GAT [37] improves GCN [18] with a node-level attention mechanism; GIN [44] generalizes GNNs in a simple but powerful way. Other examples in this category include [5, 7, 29, 38, 43, 45]. Although the graph convolution and smoothing idea in these methods works well, they tend to prefer network structure to node attributes, and thus might yield suboptimal embedding results when the node attributes contain important discriminative features.

In the third category, existing work adopts the encoder-decoder framework. For example, MVC-DNE [47] uses two autoencoders and directly copies information between hidden layers; DANE [12] adds a regularization term to constrain the distance of the two autoencoders. ANRL [52] and STNE [23] also adopt similar encoder-decoder framework. Differently, STNE takes node attributes as input and reconstructs the node neighborhood, while ANRL takes node attributes as input and reconstructs both attributes and neighborhood. Other examples in this category include [22, 24]. However, although methods in this category can provide special treatment on node attributes, it is difficult for them to flexibly take advantage of the smoothing idea from propagation based methods.

Different from these three lines of work, we design cross fusion layers to enjoy the advantages of information smoothing from propagation based methods, and the special treatment on node attributes from encoder-decoder methods; the cross fusion layers also allow flexible information exchange between them, and incorporate the local community structure during propagation.

Other Network Embedding. In addition to embedding attributed networks, embedding various other types of networks has also been studied. Examples include signed networks [16], directed networks [28, 53], dynamic networks [35, 54], heterogeneous networks [8, 11, 42, 50], multiplex networks [3, 51], etc.

6 CONCLUSIONS

In this paper, we propose an attributed network embedding approach CFane. The core of CFane is the cross fusion layers which inherit the smoothing idea from propagation based methods, encode local community structure during propagation, and provide an additional channel for treating node attributes. Experimental results demonstrate that CFane significantly improves the state-of-the-art in both node classification and node clustering. In the future, we plan to extend CFane for supervised network embedding, and explore the possibility of applying cross fusion layers to the other types of networks such as heterogeneous networks and dynamic networks.

ACKNOWLEDGMENTS

This work is supported by the Key-Area Research and Development Program of Guangdong Province (No. 2020B010164003), the National Natural Science Foundation of China (No. 61690204, 61672274), and the Collaborative Innovation Center of Novel Software Technology and Industrialization. Hanghang Tong is partially supported by NSF (1947135, 2003924, and 1939725). Yuan Yao is the corresponding author.

REFERENCES

[1] Sambaran Bandyopadhyay, N. Lokesh, and M. N. Murty. 2019. Outlier Aware Network Embedding for Attributed Networks. In *AAAI*.
[2] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. 891–900.
[3] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation Learning for Attributed Multiplex Heterogeneous Network. In *KDD*.
[4] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2018. Harp: Hierarchical representation learning for networks. In *AAAI*.
[5] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *ICLR*.
[6] Jifan Chen, Qi Zhang, and Xuanjing Huang. 2016. Incorporate group information to enhance network embedding. In *CIKM*. 1901–1904.
[7] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*. 257–266.
[8] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*.
[9] Alessandro Epasto, Silvio Lattanzi, Vahab Mirrokni, Ismail Oner Sebe, Ahmed Taei, and Sunita Verma. 2015. Ego-net community mining applied to friend suggestion. *Proceedings of the VLDB Endowment* 9, 4 (2015), 324–335.
[10] Alessandro Epasto, Silvio Lattanzi, and Renato Paes Leme. 2017. Ego-splitting framework: From non-overlapping to overlapping clusters. In *KDD*. 145–154.
[11] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1797–1806.
[12] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding.. In *IJCAI*.
[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*.
[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.
[15] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *SDM*.
[16] Junghwan Kim, Haekyu Park, Ji-Eun Lee, and U Kang. 2018. Side: representation learning in signed directed networks. In *WWW*.
[17] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).
[18] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[19] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*.
[20] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *CIKM*.
[21] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.
[22] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. 2018. Attributed social network embedding. *TKDE* (2018).
[23] Jie Liu, Zhicheng He, Lai Wei, and Yalou Huang. 2018. Content to node: Self-translation network embedding. In *KDD*.
[24] Jie Liu, Na Li, and Zhicheng He. 2019. Network Embedding with Dual Generation Tasks. In *IJCAI*.
[25] Qingqing Long, Yiming Wang, Lun Du, Guojie Song, Yilun Jin, and Wei Lin. 2019. Hierarchical Community Structure Preserving Network Embedding: A Subspace Approach. In *CIKM*. 409–418.
[26] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *JMLR* (2008).
[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.
[28] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *KDD*. 1105–1114.
[29] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-gcn: Geometric graph convolutional networks. In *ICLR*.
[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*.
[31] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*.
[32] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *KDD*.
[33] Xiaofei Sun, Jiang Guo, Xiao Ding, and Ting Liu. 2016. A General Framework for Content-enhanced Network Representation Learning. *arXiv* (2016).
[34] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*.
[35] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *ICLR*.
[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*. 5998–6008.
[37] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[38] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR*.
[39] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *KDD*.
[40] Suhang Wang, Charu Aggarwal, Jiliang Tang, and Huan Liu. 2017. Attributed signed network embedding. In *CIKM*.
[41] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. In *AAAI*.
[42] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *WWW*. 2022–2032.
[43] Felix Wu, Tianyi Zhang, Amaur Holanda de Souza, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. 2019. Simplifying graph convolutional networks. In *ICML*.
[44] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.
[45] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*. 5453–5462.
[46] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network representation learning with rich text information. In *IJCAI*.
[47] Dejian Yang, Senzhang Wang, Chaozhuo Li, Xiaoming Zhang, and Zhoujun Li. 2017. From properties to links: Deep network embedding on incomplete graphs. In *CIKM*.
[48] Hong Yang, Shirui Pan, Ling Chen, Chuan Zhou, and Peng Zhang. 2019. Low-Bit Quantization for Attributed Network Representation Learning. In *IJCAI*.
[49] Liang Yang, Zhiyang Chen, Junhua Gu, and Yuanfang Guo. 2019. Dual Self-Paced Graph Convolutional Network: Towards Reducing Attribute Distortions Induced by Topology. In *IJCAI*.
[50] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *KDD*. 793–803.
[51] Hongming Zhang, Liwei Qiu, Lingling Yi, and Yangqiu Song. 2018. Scalable Multiplex Network Embedding.. In *IJCAI*.
[52] Zhen Zhang, Hongxia Yang, Jiajun Bu, Sheng Zhou, Pinggang Yu, Jianwei Zhang, Martin Ester, and Can Wang. 2018. ANRL: Attributed Network Representation Learning via Deep Neural Networks. In *IJCAI*.
[53] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable Graph Embedding for Asymmetric Proximity. In *AAAI*. 2942–2948.
[54] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *AAAI*.