ProGAN: Network Embedding via Proximity Generative Adversarial Network

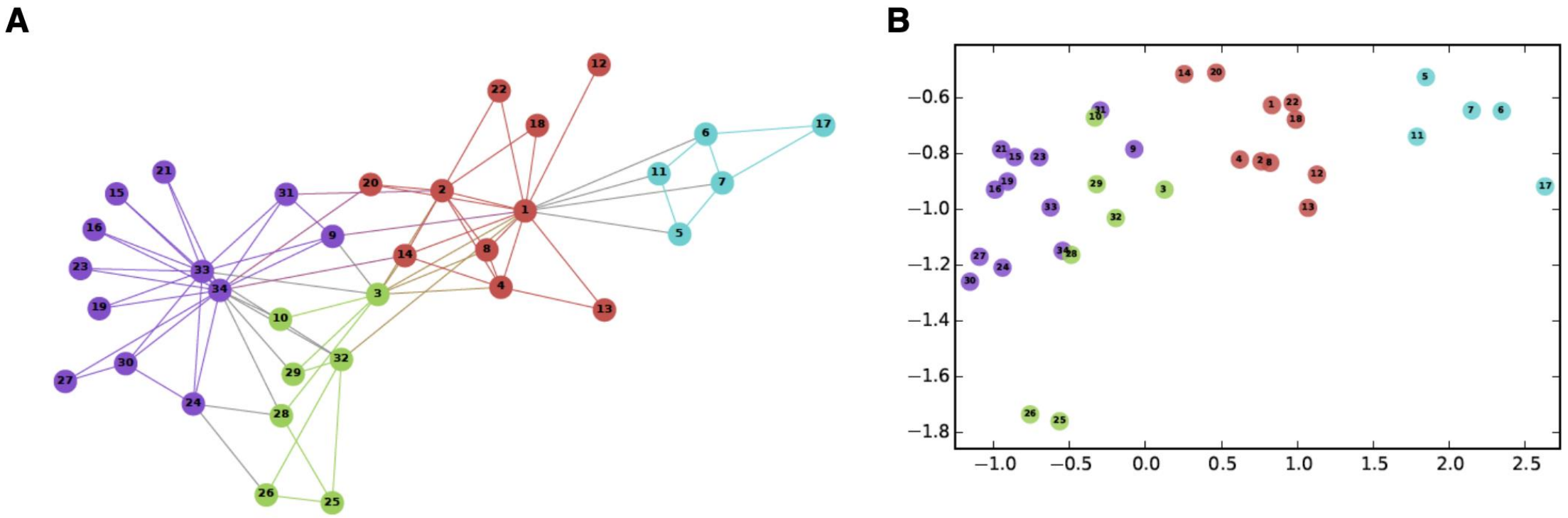
KDD’19

서울시립대학교
UNIVERSITY OF SEOUL

cheon.research@gmail.com

2020. 03. 31.

Network Embedding



https://www-cs.stanford.edu/people/jure/pubs/graphrepresentation-ieee17.pdf

Problem Definition

Let $G = \{V, W, X\}$ denote an attributed network. $V = \{v_i\}_{i=1}^{n}$ is a set of n nodes. $W = [w_{ij}] \in \Re^{n \times n}$ denotes the adjacency matrix. $w_{ij} = 1$ denotes there exists an edge between node v_i and node v_j. Otherwise, $w_{ij} = 0$. $X = [x_{ij}] \in \Re^{n \times d}$ represents the attribute matrix. The i-th row $X_{i.} \in \Re^{d}$ denotes the attribute of node v_i. In this paper, we will focus on the attributed network embedding.

Problem Definition

Definition 3.1. Network embedding is to learn a map $f : \{W, X\} \mapsto E$ where $E \in \Re^{n \times d'}$ denotes the low-dimensional representation. Meanwhile, given a triplet $\langle v_i, v_j, v_k \rangle$ in the original space such that

$$sim(v_i, v_j) > sim(v_i, v_k) , \tag{2}$$

the learned low-dimensional representation should guarantee

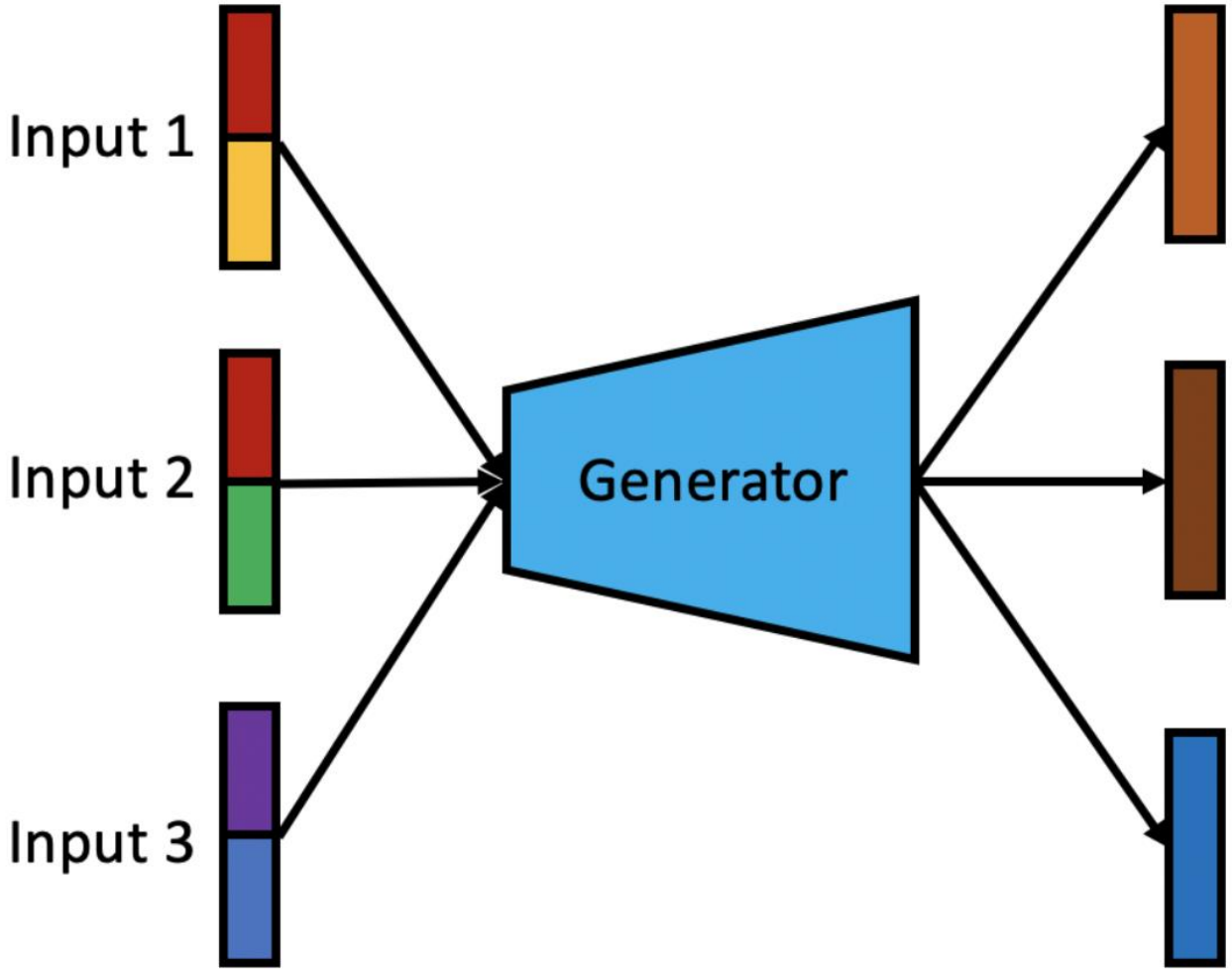
$$sim(E_i., E_j.) > sim(E_i., E_k.) , \tag{3}$$

where $sim(\cdot, \cdot)$ denotes the proximity between two data points, $E_i.$ represents the low-dimensional embedding of the i-th node v_i.

ProGAN - Generator

Generator To approximate the distribution $P(v_i, v_j, v_k)$, the generator needs to generate the triplet $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$ such that $sim(\hat{v}_i, \hat{v}_j) > sim(\hat{v}_i, \hat{v}_k)$ where $\hat{v}_i$ denotes the generated nodes. But how to generate this kind of triplets? Here, we consider a generator $\hat{v} \sim G(z_1, z_2)$ where z_1 corresponds to the first input noise while z_2 corresponds to the second input noise. Comparing with the standard GAN which has only one input noise z, our proposed ProGAN decouples the input noise into two parts: z_1 and z_2. In this way, our objective is to learn the generator such that varying z_1 or z_2 can control the similarity of two generated nodes. Doing so allows us to generate the desired triplets.

ProGAN - Generator



ProGAN - Generator

$$\mathcal{L}_G = E_{z_1 \sim p(z), z_2 \sim p(z)}[\log(D_1(G(z_1, z_2)))] + E_{z_1 \sim p(z), z_2 \sim p(z), z_2' \sim p(z)}[\log \sigma(s_1)] + E_{z_1 \sim p(z), z_2 \sim p(z), z_1'' \sim p(z), z_2'' \sim p(z)}[\log(1 - \sigma(s_2))]$$

$$s_1 = D_2(G(z_1, z_2))^T D_2(G(z_1, z_2'))$$

similarity

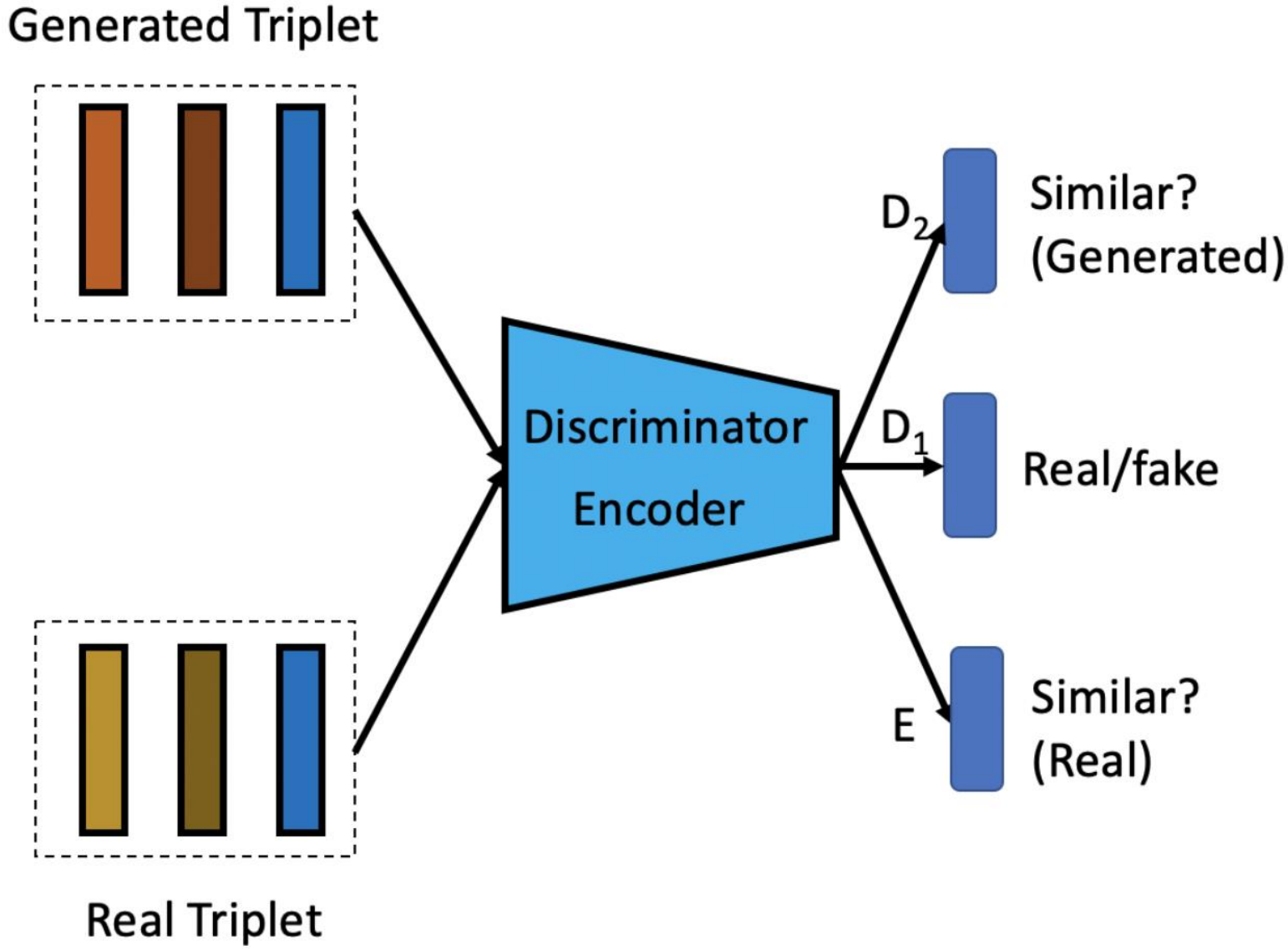
$$s_2 = D_2(G(z_1, z_2))^T D_2(G(z_1'', z_2''))$$

dissimilarity

ProGAN - Discriminator

For the discriminator, we need real nodes so that we can distinguish the real and generated ones. But how to represent them? Here, in this paper, we employ the node attribute $X_i.$ to represent the real node v_i. Then, the triplet can be represented in the same way. Correspondingly, the generator should generate node attributes $\hat{X}_i..$ In other words, to fool the discriminator, the generator will learn a distribution $Q(\hat{X}_i.)$ to approximate the real node distribution $P(X_i.)$. Furthermore, the discriminator will also guide the generator to learn a distribution $Q(\hat{X}_i., \hat{X}_j., \hat{X}_k.)$ to approximate $P(X_i., X_j., X_k.)$.

ProGAN - Discriminator



ProGAN - Discriminator

$$\mathcal{L}_{D_1} = E_{x\sim p(x)}[\log D_1(x)] + E_{z_1\sim p(z), z_2\sim p(z)}[\log(1 - D_1(G(z_1, z_2)))] ,$$

$$\mathcal{L}_{D_2} = E_{z_1\sim p(z), z_2\sim p(z), z_2'\sim p(z)}[\log \sigma(s_1)] + E_{z_1\sim p(z), z_2\sim p(z), z_1''\sim p(z), z_2''\sim p(z)}[\log(1 - \sigma(s_2))]$$

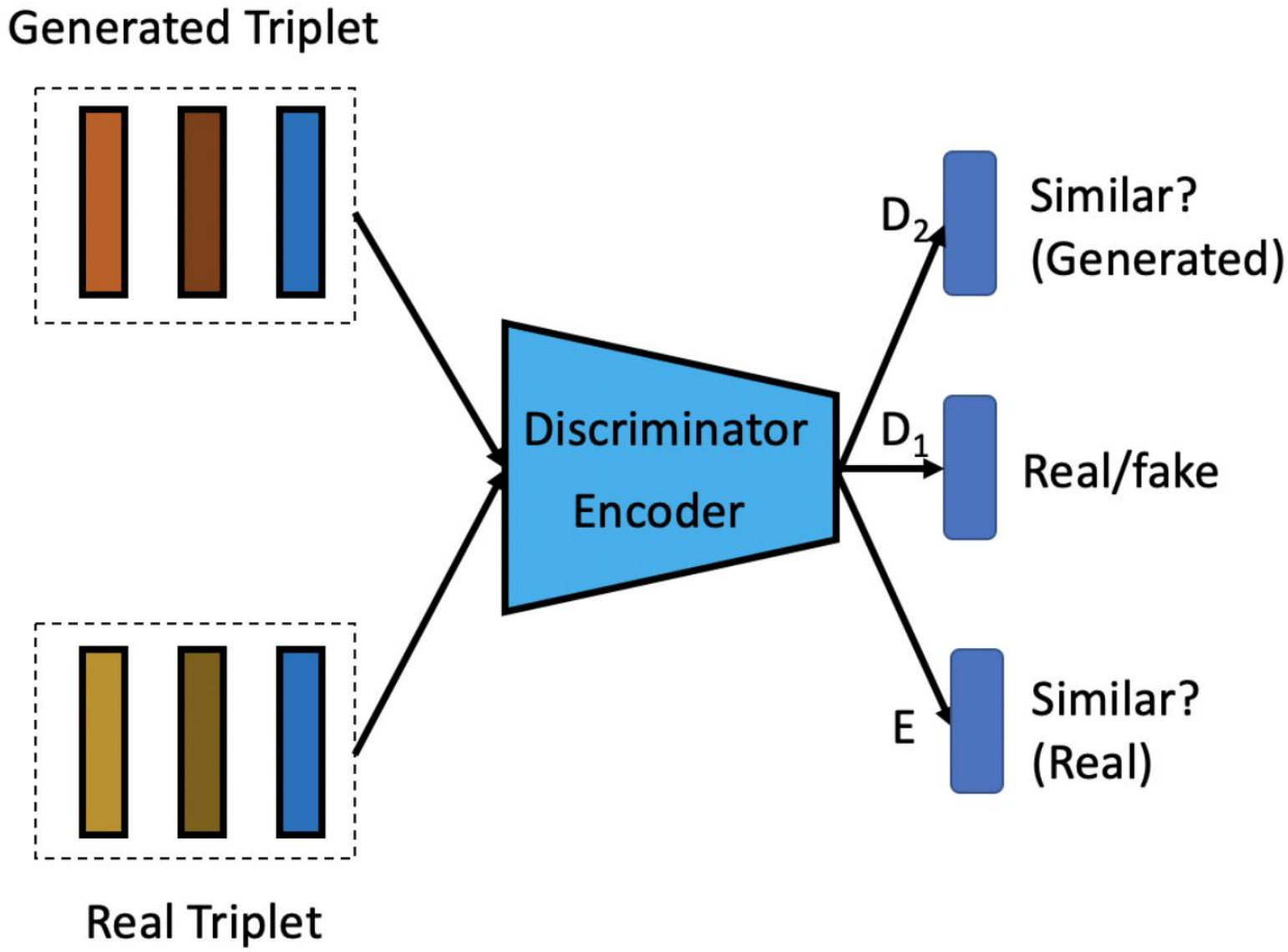
$$s_1 = D_2(G(z_1, z_2))^T D_2(G(z_1, z_2'))$$ **similarity**

$$s_2 = D_2(G(z_1, z_2))^T D_2(G(z_1'', z_2''))$$ **dissimilarity**

ProGAN - Encoder

For the discovered proximity, we also resort to the triplet. Specifically, to construct the real triplet $\langle X_i., X_j., X_k. \rangle$ such that $sim(X_i., X_j.) > sim(X_i., X_k.)$, we use the following steps. At first, we randomly select the reference node $X_i.$. Then, $X_j.$ is selected from $\{j|w_{ij} = 1\}$ while $X_k.$ is selected from $\{k|w_{ik} = 0\}$. In other words, if there exists an edge between two nodes, they are similar. Otherwise, they are dissimilar. For the underlying proximity, we directly utilize the generated triplet $\langle \hat{X}_i., \hat{X}_j., \hat{X}_k. \rangle$ such that $sim(\hat{X}_i., \hat{X}_j.) > sim(\hat{X}_i., \hat{X}_k.)$. With these proximities, we expect $sim(E_i., E_j.) > sim(E_i., E_k.)$ and $sim(\hat{E}_i., \hat{E}_j.) > sim(\hat{E}_i., \hat{E}_k.)$ in the low-dimensional space. In this way, we can push similar nodes together and push away dissimilar nodes in the low-dimensional space.

ProGAN - Encoder



ProGAN - Encoder

$$\mathcal{L}_E = E_{(x_1, x_2, x_3) \sim p(x,y,z)}[\log \sigma(E(x_1)^T E(x_2)) + \log(1 - \sigma(E(x_1)^T E(x_3)))] + E_{z_1 \sim p(z), z_2 \sim p(z), z_2' \sim p(z)}[\log \sigma(t_1)] + E_{z_1 \sim p(z), z_2 \sim p(z), z_1'' \sim p(z), z_2'' \sim p(z)}[\log(1 - \sigma(t_2))]$$

$$t_1 = E(G(z_1, z_2))^T E(G(z_1, z_2'))$$

similarity

$$t_2 = E(G(z_1, z_2))^T E(G(z_1'', z_2''))$$

dissimilarity

Algorithm

Algorithm 1 Algorithm to optimize ProGAN.

1: **repeat**
2: Sample the input noise as Eq.(5) to optimize the generator loss $\mathcal{L}_G$.
3: Sample the input noise and real nodes as Eq.(4) to optimize the descriminator loss $\mathcal{L}_{D_1} + \mathcal{L}_{D_2}$.
4: Sample the real triplet and the input noise as Eq.(6) to optimize the descriminator loss $\mathcal{L}_E$.
5: **until** Converges

Datasets

Dataset	# Nodes	#Edges	#Attributes	#Labels
Citeseer	3,312	4,660	3,703	6
Cora	2,708	5,278	1,433	7
Flickr	7,564	239,365	12,047	9
Blogcatalog	5,196	171,743	8,189	6

Experiments

❑ Node classification result of Citeseer dataset

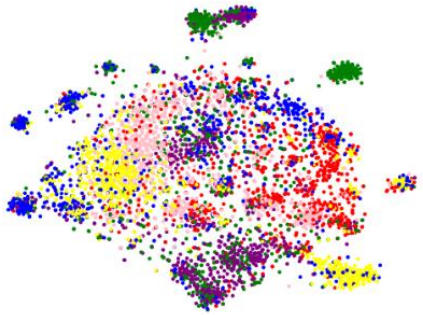
Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.5146	0.4604	0.5623	0.5149	0.5830	0.5397
Node2Vec	0.5059	0.4541	0.5744	0.5249	0.5812	0.5339
LINE	0.4951	0.4472	0.5317	0.4778	0.5395	0.4942
GraRep	0.4908	0.4355	0.5326	0.4622	0.5335	0.4662
GraphGAN	0.4260	0.3837	0.5347	0.4888	0.5570	0.5146
TADW	0.6451	0.5990	0.7055	0.6487	0.7174	0.6639
GAE	0.6273	0.5806	0.6727	0.6055	0.6868	0.6059
SAGE	0.5039	0.4707	0.5692	0.5305	0.5999	0.5563
DANE	0.6585	0.6121	0.7085	0.6528	0.7115	0.6553
ProGAN	**0.7186**	**0.6488**	**0.7417**	**0.6748**	**0.7440**	**0.6931**

Experiments

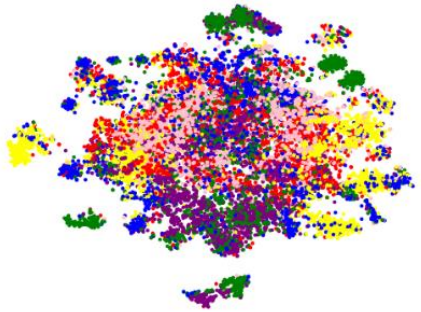
❑ Node classification result of Citeseer dataset

Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.7424	0.7368	0.7975	0.7866	0.8148	0.8032
Node2Vec	0.7777	0.7649	0.8107	0.8001	0.8118	0.8007
LINE	0.7473	0.7399	0.7943	0.7883	0.8081	0.8011
GraRep	0.7609	0.7510	0.7700	0.7558	0.7764	0.7617
GraphGAN	0.6957	0.6804	0.7405	0.7241	0.7668	0.7557
TADW	0.7683	0.7462	0.8201	0.7989	0.8435	0.8293
GAE	0.7662	0.7587	0.7980	0.7852	0.8015	0.7896
SAGE	0.6608	0.6403	0.7664	0.7520	0.8044	0.7921
DANE	0.7769	0.7558	0.8212	0.8062	0.8258	0.8094
ProGAN	**0.8080**	**0.7866**	**0.8365**	**0.8172**	**0.8486**	**0.8357**

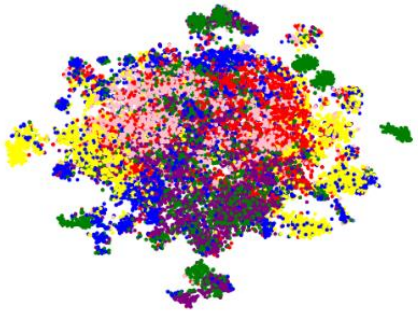
Experiments



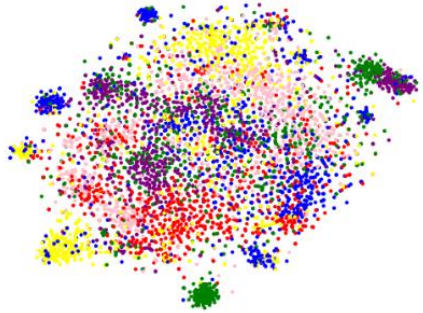
(a) DeepWalk



(b) GraRep

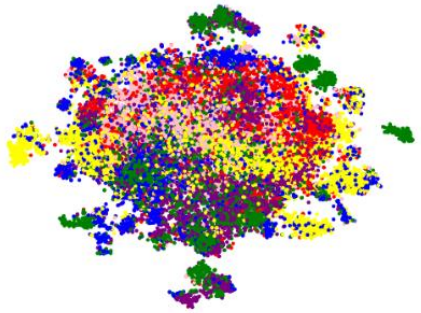


(c) LINE

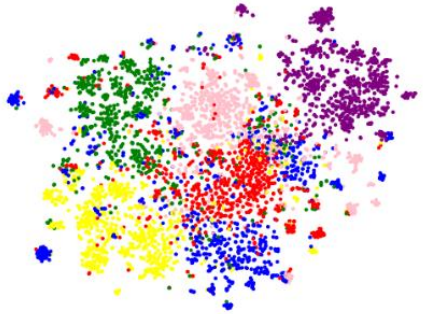


(d) GraphGAN

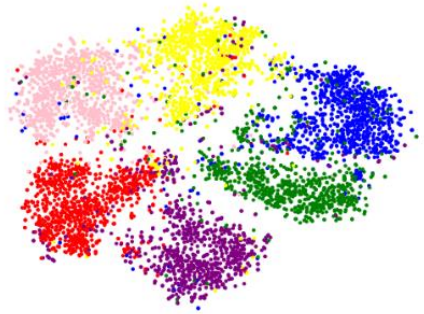
(e) TADW



(f) SAGE



(g) DANE



(h) ProGAN