ProGAN: Network Embedding via Proximity Generative Adversarial Network

Hongchang Gao
Electrical & Computer Engineering
University of Pittsburgh
Pittsburgh, USA
JD Finance America Corporation
hongchanggao@gmail.com

Jian Pei
School of Computing Science
Simon Fraser University
Burnaby, Canada
JD.com
jpei@cs.sfu.ca

Heng Huang*
Electrical & Computer Engineering
University of Pittsburgh
Pittsburgh, USA
JD Finance America Corporation
heng.huang@pitt.edu

ABSTRACT

Network embedding has attracted increasing attention in recent few years, which is to learn a low-dimensional representation for each node of a network to benefit downstream tasks, such as node classification, link prediction, and network visualization. Essentially, the task of network embedding can be decoupled into *discovering* the proximity in the original space and *preserving* it in the low-dimensional space. Only with the well-discovered proximity can we preserve it in the low-dimensional space. Thus, it is critical to discover the proximity between different nodes to learn good node representations. To address this problem, in this paper, we propose a novel proximity generative adversarial network (ProGAN) which can generate proximities. As a result, the generated proximity can help to discover the complicated underlying proximity to benefit network embedding. To generate proximities, we design a novel neural network architecture to fulfill it. In particular, the generation of proximities is instantiated to the generation of triplets of nodes, which encodes the similarity relationship between different nodes. In this way, the proposed ProGAN can generate proximities successfully to benefit network embedding. At last, extensive experimental results have verified the effectiveness of ProGAN.

CCS CONCEPTS

• **Computing methodologies → Machine learning approaches**; Neural networks.

KEYWORDS

Nework Embedding; Generative Adversarial Networks; Proximity

ACM Reference Format:
Hongchang Gao, Jian Pei, and Heng Huang. 2019. ProGAN: Network Embedding via Proximity Generative Adversarial Network. In *Anchorage '19: KDD, August 4-8, 2019, Anchorage, Alaska.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

*To whom all correspondence should be addressed. This work was partially supported by U.S. NSF IIS 1836945, IIS 1836938, DBI 1836866, IIS 1845666, IIS 1852606, IIS 1838627, IIS 1837956.

1 INTRODUCTION

Networks are ubiquitous in real-world web applications, such as social network, hyperlink network, and so on. Analyzing networks for desired tasks is becoming more and more important. As a fundamental tool to analyze networks, network embedding has attracted a surge of attention in recent few years. Essentially, network embedding is to learn a low-dimensional representation for each node in a network. This low-dimensional node representation can preserve the proximity between different nodes of the network. Then, downstream tasks, such as node classification, link prediction, and network visualization, can benefit from this representation.

In recent years, numerous network embedding methods have been proposed, such as DeepWalk [19], Node2Vec [9], LINE [21], GraRep [2], and Struc2Vec [20]. The basic idea of these methods is to make the low-dimensional representation preserve the proximity between different nodes in a network as good as possible. Thus, network embedding methods include two phases. The first phase is to *discover* the proximity between different nodes in a network. The second phase is to *preserve* the proximity in the low-dimensional space. Only with the well-discovered proximity can we preserve it in the low-dimensional space. Thus, it is considerably important to discover the underlying proximity in a network well.

To discover the underlying proximity in a network, various proximities have been explored in recent years. Specifically, these proximities include the first-order proximity, second-order proximity, high-order proximity, and so on. In detail, the first-order proximity [21] considers that two nodes are similar if there is an edge between them. However, the discovered edges in a network are usually very sparse so that they are not enough to disclose the proximity between different nodes. The second-order proximity [21] views two nodes similar if they share similar neighbors. Thus, the second-order proximity is supposed to discover more underlying relationships than the first-order proximity. Furthermore, GraRep [2] proposes to discover the high-order proximity by constructing the k-step probability transition matrix explicitly. In this way, the long-distance relationship between two different nodes can be captured. Additionally, DeepWalk [19] employs random walk to discover the high-order proximity. Specifically, it utilizes random walk to get node sequences. For each node sequence, it uses a sliding window to get the neighborhood of its nodes. As a result, the high-order proximity information can be captured [24]. In this paper, we will propose a novel method to discover the underlying proximity among different nodes to benefit network embedding.

In the past few years, generative adversarial networks (GAN) [8] have shown promising results in a wide variety of tasks, such

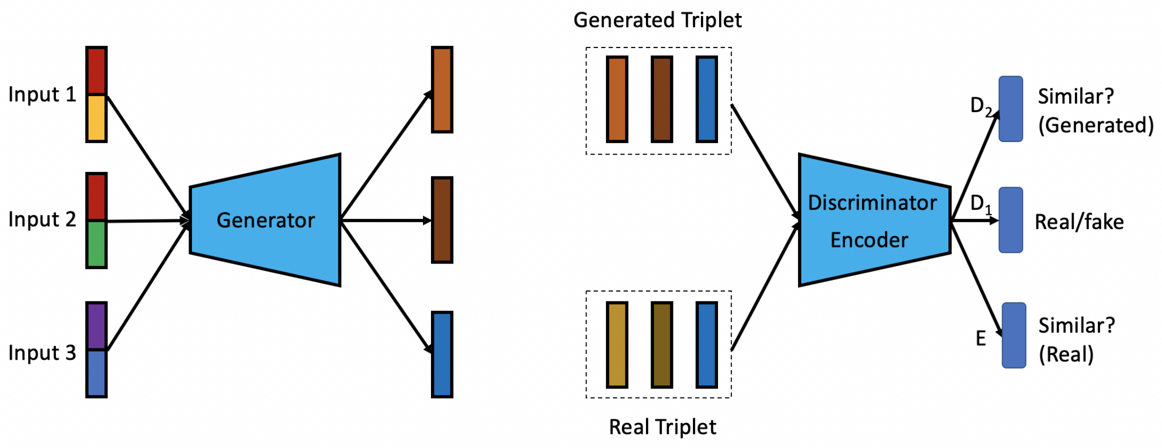


Figure 1: The architecture of our proposed ProGAN. For the generator, Inputs 1 and 2 share a common part so that they generate similar output, while Input 3 does not share any parts with the other two so that it generates different output. For the discriminator and encoder, the inputs are real and generated triplets. The discriminator distinguishes real and fake nodes, and discriminates whether two generated nodes are similar. The encoder discriminates whether two real nodes are similar.

as image generation [26], text generation [25]. The basic idea of GAN is to learn a map which can transform a noise from a simple distribution to a sample from a complicated distribution. This map is usually constructed by a deep neural network which has large expressivity. Inspired by the development of GAN, we propose to use GAN to generate proximities between different nodes to approximate the underlying proximity. In this way, the generated proximity can capture the underlying relationship between different nodes, which will benefit the network embedding. However, the standard GAN is used to generate samples rather than proximities. Thus, how to generate the proximity is challenging.

To address the aforementioned problems, in this paper, we propose a novel proximity generative adversarial network (ProGAN) for network embedding. Specifically, to capture the underlying proximity, ProGAN tries to generate proximities to approximate the distribution of the real proximity. In this way, undiscovered proximities can be captured by the generated ones. Furthermore, to generate the proximity, we instantiate it to the relationship of a triplet of nodes, in which the positive node is similar to the reference node while the negative node is dissimilar with that. By generating this kind of triplets, ProGAN can capture the underlying proximity. Moreover, to generate the desired triplet, we propose a novel neural network architecture as shown in Figure 1. With this novel architecture, ProGAN can generate similar nodes from similar input noise and dissimilar nodes from dissimilar input noise. With the generated and real triplets, ProGAN trains an encoder to learn node representations, which can preserve both the real proximity and the generated proximity. At last, extensive experimental results have verified the performance of our proposed ProGAN.

2 RELATED WORKS

As a fundamental tool to analyze networks, various network embedding methods have been proposed in recent years. In this part, we will give a brief review on these related works.

2.1 Network Embedding

In terms of the availability of node attributes, existing methods can be categorized into two classes: *pure network embedding* and *attributed network embedding*. Plain network embedding methods [2, 9, 19–21] only employ the topological structure to learn node representations, while attributed network embedding methods [5, 10–13, 18, 24] utilize both the topological structure and node attributes.

Pure Network Embedding The seminal DeepWalk [19] formulates network embedding as word embedding. Specifically, it employs random walk on the network to get node sequences which can be viewed as sentences. With these node sequences, DeepWalk then utilizes the Skip-Gram model to learn node representations. Later, Node2Vec [9] proposes a biased random walk method, which combines the breadth-first and depth-first sampling to preserve the local and global proximity, to construct the context of nodes. Afterwards, LINE [21] is proposed to preserve the first-order and second-order proximity when learning the node representation. GraRep [2] aims to preserve the high-order proximity. Specifically, two nodes having similar k-step neighbors should have similar latent representations. However, all these methods ignore node attributes when learning the low-dimensional node representation.

Attributed Network Embedding In an attributed network, nodes usually possess rich information. By exploring such kind of rich attributed information, the proximity among different nodes can be discovered and preserved better when learning node representations. For instance, [24] proposes to incorporate both topological structure and node attributes by using inductive matrix factorization method. [11] proposes to factorize the attribute matrix and regularize the factorization by the toplogical structure. Recently, [5] utilizes a multi-modal auto-encoder to combine topological structure and node attributes to learn node representations. By incorporating node attributes, these methods have shown some improvement over the counterpart which only employs the topological structure.

However, most of these methods are shallow methods, failing to capture the highly non-linear property in a network. Networks in real-world applications are usually complicated and highly non-linear. Thus, it is important to capture the high non-linearity in the network.

2.2 Deep Network Embedding

Recent developments of deep neural networks have accelerated much progress in data mining and machine learning. Deep neural networks have much larger expressivity to capture the highly non-linear property of the data than shallow methods. An example is the seminal work [15] which adopts deep convolutional neural network to learn image representation for classification and achieves significant improvement over traditional methods. From then on, deep neural network has been widely used in a wide variety of applications.

In recent years, some works [1, 3, 5–7, 10, 13, 22] have been proposed to apply deep neural networks for network embedding. For example, [3] proposes a heterogeneous network embedding method. This method focuses on heterogeneous data embedding, such as image-text embedding or text-text embedding, ignoring to explore the network structure. The work [22] employs deep neural network for network embedding, in which an autoencoder is utilized to learn the low-dimensional representation for each node. This model can discover the highly non-linear structure in the network while preserving the similarity. In addition, [13] proposes a semi-supervised network embedding method based on graph convolutional neural network by combining the topological structure and the node attribute. Recently, [1] utilizes a multi-layer perceptron neural network to map each node in the network to a Gaussian distribution, which can capture the uncertainty of the learned representation.

Recently, a variant of deep neural networks, generative adversarial network (GAN) [8], has attracted much attention due to its impressing performance on the unsupervised task. Its basic idea is to learn a map which can transform a simple distribution to a complicated distribution. Specifically, GAN includes a generator and a discriminator. The generator tries to generate samples as real as possible while the discriminator is to distinguish the generated samples and the real ones. Formally, the objective function of GAN is defined as follows:

$$\min_{\phi} \max_{\theta} E_{x \sim p(x)}[\log D_\theta(x)] + E_{z \sim p(z)}[\log(1 - D_\theta(G_\phi(z)))] , \quad (1)$$

where $p(x)$ denotes the real data distribution and $p(z)$ represents the simple prior distribution. $D_\theta(\cdot)$ corresponds to the discriminator while $G_\phi(\cdot)$ corresponds to the generator. By optimizing this objective function, the generator $G_\phi(\cdot)$ can learn a map to transform the prior distribution $p(z)$ to the complicated data distribution $q(x)$ where $q(x)$ is the approximation to the real data distribution $p(x)$.

Due to the impressing performance of GAN, some researchers have tried to apply GAN for network embedding. For instance, [23] proposes GraphGAN to learn low-dimensional node representations. Specifically, network embedding is to push similar nodes together and push away dissimilar nodes. When pushing away dissimilar nodes, it needs to sample negative nodes for the reference node. GraphGAN employs a generator to generate the sampling distribution to sample the negative nodes, benefiting the embedding result. [4] proposes the adversarial network embedding, which utilizes the adversarial technique to regularize the learned representation. In this paper, we will exploit GAN to discover the underlying proximity to benefit node embedding.

3 NETWORK EMBEDDING VIA PROXIMAL GENERATIVE ADVERSARIAL NETWORK

3.1 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 $X_{i\cdot} \in \Re^d$ denotes the attribute of node v_i. In this paper, we will focus on the attributed network embedding.

Network embedding is to learn a low-dimensional representation for each node v_i from the topological structure W and attributes X. Additionally, the low-dimensional representation should preserve the proximity between different nodes. In other words, in the low-dimensional space, a network embedding method should push similar nodes together and push dissimilar nodes away. Therefore, the proximity between different nodes can be instantiated to the triplet $\langle v_i, v_j, v_k \rangle$ where v_i denotes the reference node, v_j represents the positive node which is similar with v_i, v_k stands for the negative node which is dissimilar with v_i. Therefore, we have the following formal 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

$$sim(v_i, v_j) > sim(v_i, v_k) , \quad (2)$$

the learned low-dimensional representation should guarantee

$$sim(E_{i\cdot}, E_{j\cdot}) > sim(E_{i\cdot}, E_{k\cdot}) , \quad (3)$$

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

This definition indicates that the task of network embedding can be decoupled into *discovering* the proximity in the original space and *preserving* it in the low-dimensional space. Only with the well-discovered proximity can we preserve it in the low-dimensional space. Thus, discovering the proximity between different nodes is critical to learn a good node representation. However, the edges in a network are very sparse so that they are not enough to disclose the proximity between different nodes. Existing network embedding algorithms proposed various methods to discover underlying proximities, such as first-order proximity, second-order proximity, and so on. However, most of them fail to fully utilize both the topological structure and node attributes. To address these problems, we will propose a novel proximity generative adversarial network to discover the underlying proximity among different nodes, benefiting network embedding.

3.2 Proximity Generative Adversarial Network

Unlike existing methods which discover proximities among different nodes, we propose to generate the proximity. But how to generate it is challenging. Inspired by the development of generative adversarial networks, we propose to employ GAN to generate the underlying proximity. However, the standard GAN is designed to generate samples rather than proximities. Thus, how to generate the underlying proximity is also challenging. To address these challenges, we propose a novel proximity generative adversarial network (ProGAN).

As discussed in the previous subsection, the proximity can be instantiated to the triplet $\langle v_i, v_j, v_k \rangle$ such that $sim(v_i, v_j) > sim(v_i, v_k)$. Thus, the key idea of generating the proximity is to generate triplets. Formally, assume the true distribution of the triplet is denoted by $P(v_i, v_j, v_k)$ which is unknown. Our task is to learn a distribution $Q(v_i, v_j, v_k)$ to approximate $P(v_i, v_j, v_k)$. Since $Q(v_i, v_j, v_k)$ is similar with $P(v_i, v_j, v_k)$, the generated triplets from $Q(v_i, v_j, v_k)$ can disclose the real proximity. Thus, by generating triplets, ProGAN can discover the underlying complicated proximity between different nodes. In addition, when generating the triplet to get the proximity, it is supposed that the generated individual nodes in the triplet are as real as the true nodes. In other words, our another task is to learn the distribution $Q(v_i)$ to approximate the true node distribution $P(v_i)$. Moreover, after discovering the underlying proximity, we should preserve it when learning the low-dimensional representation. To this end, ProGAN includes three components: generator, discriminator, and encoder. Generator is expected to generate the desired triplet and node. Discriminator needs to discriminate them from the real ones. Encoder is to learn the low-dimensional representation for each node by preserving the real and generated proximity. In the following part, we will describe the detail of these three components.

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.

In detail, to generate two similar nodes $(\hat{v}_i, \hat{v}_j)$, ProGAN enforces their input noise to share a common z_1. Otherwise, there is no such a constraint. Thus, to generate the desired triplet $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$ such that $sim(\hat{v}_i, \hat{v}_j) > sim(\hat{v}_i, \hat{v}_k)$, the input to the generator is (z_{i_1}, z_{i_2}), (z_{i_1}, z_{j_2}), and (z_{k_1}, z_{k_2}) respectively. Apparently, $\hat{v}_i$ and $\hat{v}_j$ share the same input noise z_{i_1}, then they are expected to be similar with each other. In addition, $\hat{v}_i$ and $\hat{v}_j$ does not share the second input noise, then they will not be exactly same with each other. On the other hand, $\hat{v}_i$ and $\hat{v}_k$ do not share any common input, thus they are not expected to be similar.

On the other hand, to approximate the distribution $P(v_i)$, we employ the same generator $G(z_1, z_2)$. When concatenating the two input z_1 and z_2 together, this generator acts as the standard one in GAN to generate nodes. Thus, with this proposed generator $G(z_1, z_2)$, we can generate not only the desired node $\hat{v}_i$ but also the triplet $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$. Since the generated nodes are supposed to be similar with real nodes and the generated triplet $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$ is also similar to real triplets, the proximity between generated nodes should also be similar with real nodes. Thus, by generating triplets $\langle \hat{v}_i, \hat{v}_j, \hat{v}_k \rangle$, we can discover the underlying proximity between different nodes.

Discriminator To learn the aforementioned generator, the standard discriminator of GAN is not enough, since it cannot disentangle the latent space. As a result, there is no possibility to control the similarity of generated nodes. The discriminator of ProGAN is supposed to complete two goals. The first goal is to distinguish the generated nodes and the real ones. This is same as the standard GAN, which will guide the generator to generate nodes as real as possible. The second goal is to determine whether the generated triplet satisfies $sim(\hat{v}_i, \hat{v}_j) > sim(\hat{v}_i, \hat{v}_k)$. In other words, the discriminator should determine whether two generated nodes are similar or not. In this way, the generated triplet can encode the proximity information. As a result, the discriminator will guide the generator to approximate two distributions $P(v_i)$ and $P(v_i, v_j, v_k)$.

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 $X_{i\cdot}$ to represent the real node v_i. Then, the triplet can be represented in the same way. Correspondingly, the generator should generate node attributes $\hat{X}_{i\cdot}$. In other words, to fool the discriminator, the generator will learn a distribution $Q(\hat{X}_{i\cdot})$ to approximate the real node distribution $P(X_{i\cdot})$. Furthermore, the discriminator will also guide the generator to learn a distribution $Q(\hat{X}_{i\cdot}, \hat{X}_{j\cdot}, \hat{X}_{k\cdot})$ to approximate $P(X_{i\cdot}, X_{j\cdot}, X_{k\cdot})$.

Encoder After obtaining the proximity between different nodes, we can use it to learn the low-dimensional representation for each node. To do that, the proximity should be preserved in the low-dimensional space. As we discussed early, the discovered proximity is not enough to learn a good representation. Thus, we should utilize both the discovered proximity and the underlying one.

For the discovered proximity, we also resort to the triplet. Specifically, to construct the real triplet $\langle X_{i\cdot}, X_{j\cdot}, X_{k\cdot} \rangle$ such that $sim(X_{i\cdot}, X_{j\cdot}) > sim(X_{i\cdot}, X_{k\cdot})$, we use the following steps. At first, we randomly select the reference node $X_{i\cdot}$. Then, $X_{j\cdot}$ is selected from $\{j | w_{ij} = 1\}$ while $X_{k\cdot}$ 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 $\langle \hat{X}_{i\cdot}, \hat{X}_{j\cdot}, \hat{X}_{k\cdot} \rangle$ such that $sim(\hat{X}_{i\cdot}, \hat{X}_{j\cdot}) > sim(\hat{X}_{i\cdot}, \hat{X}_{k\cdot})$. With these proximities, we expect $sim(E_{i\cdot}, E_{j\cdot}) > sim(E_{i\cdot}, E_{k\cdot})$ and $sim(\hat{E}_{i\cdot}, \hat{E}_{j\cdot}) > sim(\hat{E}_{i\cdot}, \hat{E}_{k\cdot})$ in the low-dimensional space. In this way, we can push similar nodes together and push away dissimilar nodes in the low-dimensional space.

Since the second goal of the discriminator is also push similar generated nodes together and push away dissimilar generated nodes, the encoder can share the same architecture with the discriminator. The only difference is that the discriminator has one more function to distinguish the real nodes and the generated ones. Therefore, in this paper, we use a single neural network for both the discriminator and encoder. Based on this common neural network, we add one more branch at the last layer for the discriminator to differentiate the generated nodes from the real ones.

At last, all of these three components are implemented by deep neural networks to capture the highly non-linear property in a network. In summary, the generator can generate the desired triplet to discover the underlying proximity. The discriminator tries to guide the generator to generate nodes and triplets to be similar as real ones. With the discovered proximity and the synthesized underlying proximity, the encoder employs a deep neural network to learn low-dimensional node representations. Comparing with existing methods, the difference lies in that we employ our proposed ProGAN to synthesize the underlying proximity rather than construct it by heuristic strategies.

Loss Function To learn the three components in ProGAN, we are going to define the objective function. Overall, it includes the *adversarial loss* and *encoder loss*. The adversarial loss will enforce ProGAN to generate nodes and triplets as real as possible and the encoder loss will push ProGAN to learn a good node representation.

Regarding the adversarial loss, it is designed as a minimax game between the discriminator and the generator, in which the discriminator is trained to distinguish the real and generated nodes while the generator is trained to generate nodes to fool the discriminator. Specifically, the discriminator's loss function is defined as follows:

$$\begin{aligned}\mathcal{L}_{D_1} &= E_{x\sim p(x)}[\log D_1(x)] \\ &\quad + E_{z_1\sim p(z), z_2\sim p(z)}[\log(1 - D_1(G(z_1, z_2)))] , \\ \mathcal{L}_{D_2} &= E_{z_1\sim p(z), z_2\sim p(z), z_2'\sim p(z)}[\log \sigma(s_1)] \\ &\quad + E_{z_1\sim p(z), z_2\sim p(z), z_1''\sim p(z), z_2''\sim p(z)}[\log(1 - \sigma(s_2))] ,\end{aligned} \tag{4}$$

where $s_1 = D_2(G(z_1, z_2))^T D_2(G(z_1, z_2'))$ measures the similarity of two similar nodes while $s_2 = D_2(G(z_1, z_2))^T D_2(G(z_1'', z_2''))$ measures that of two dissimilar nodes. In addition, $\sigma(\cdot)$ denotes the Sigmoid function. Note that the discriminator has two branches as shown in the Figure 1. Here, we use D_1 and D_2 to denote the output of these two branches respectively. Correspondingly, $\mathcal{L}_{D_1}$ implements the first goal of the discriminator to distinguish the real and generated nodes. $\mathcal{L}_{D_2}$ fulfills the second goal to discriminate whether two generated nodes are similar or not. As discussed earlier, $G(z_1, z_2)$ and $G(z_1, z_2')$ share the same input z_1 so that they are considered to be similar. On the contrary, $G(z_1, z_2)$ and $G(z_1'', z_2'')$ do not share any input so that they are supposed to be dissimilar.

Regarding the generator loss, it should guide the generator to fool the discriminator and synthesize similar nodes when given similar input. To this end, the loss function of the generator is defined as follows:

$$\begin{aligned}\mathcal{L}_G &= E_{z_1\sim p(z), z_2\sim p(z)}[\log(D_1(G(z_1, z_2)))] \\ &\quad + E_{z_1\sim p(z), z_2\sim p(z), z_2'\sim p(z)}[\log \sigma(s_1)] \\ &\quad + E_{z_1\sim p(z), z_2\sim p(z), z_1''\sim p(z), z_2''\sim p(z)}[\log(1 - \sigma(s_2))] ,\end{aligned} \tag{5}$$

where s_1 and s_2 are defined as the discriminator. Specifically, the first term guides the generator to generate nodes as real as possible. The second and third terms guide the generator to generate similar nodes when given similar input, while generate dissimilar nodes when given different input.

Regarding the encoder loss, it should push similar nodes together and push dissimilar nodes away. In addition, to train the encoder, we utilize both the discovered proximity and the underlying proximity generated by the generator. Therefore, the loss function of the encoder is defined as follows:

$$\begin{aligned}\mathcal{L}_E &= E_{(x_1, x_2, x_3)\sim p(x, y, z)}[\log \sigma(E(x_1)^T E(x_2)) \\ &\quad + \log(1 - \sigma(E(x_1)^T E(x_3)))] \\ &\quad + E_{z_1\sim p(z), z_2\sim p(z), z_2'\sim p(z)}[\log \sigma(t_1)] \\ &\quad + E_{z_1\sim p(z), z_2\sim p(z), z_1''\sim p(z), z_2''\sim p(z)}[\log(1 - \sigma(t_2))] ,\end{aligned} \tag{6}$$

where $t_1 = E(G(z_1, z_2))^T E(G(z_1, z_2'))$ measures the similarity of node representations of two similar nodes while that of two dissimilar nodes is measured by $t_2 = E(G(z_1, z_2))^T E(G(z_1'', z_2''))$. Here, the first two terms act on the discovered proximity while the last two terms act on the underlying proximity generated by the generator. Note that the parameter of encoder E is same as the discriminator D_2. Thus, we can train the encoder and the discriminator simultaneously.

By optimizing these objective functions, we can learn the parameter of three components of our ProGAN. Particularly, we can use the stochastic gradient descent method to optimize it easily. At last, we summarize the optimization step in Algorithm 1. Note that step 3 and 4 can be updated simultaneously.

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
```

4 EXPERIMENTS

In this section, we will conduct extensive experiments to show the performance of our proposed methods.

4.1 Dataset Descriptions

Throughout this paper, four attributed networks are employed, including citation networks, social networks. The details about these datasets are described as follows.

- Citeseer [17] is a paper citation network. Papers from different topics constitute nodes and the citation among them composes edges. We use the content of papers as node attributes and topics as class labels.
- Cora [17] is also a paper citation network. Similarly, nodes are papers from different topics, edges are the citation among different papers, and the node attribute is the bag-of-words representation of the corresponding paper. In addition, the topics that papers belong to are considered as class labels.
- Flickr [11] is a social network where users share their photos. Users follow each other to form a network. Here, the tags on their images are used as the attribute. Additionally, the groups that users joined are considered as class labels.
- Blogcatalog [11] is also a social network where nodes are users and edges are the friendship between different users. The attribute of nodes is the extracted keywords from their

Table 1: Descriptions of Benchmark 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

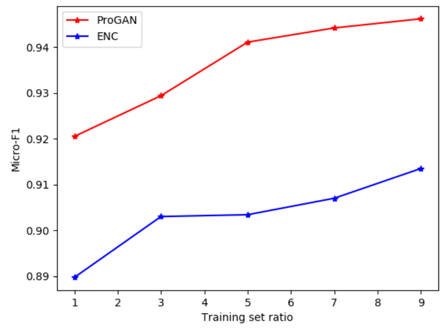
blogs. Regarding the class label, we use the predefined categories that blogs belong to.

For all datasets, we normalize its attribute to have a unit length. At last, we summarize the statistics of these networks in Table 1.

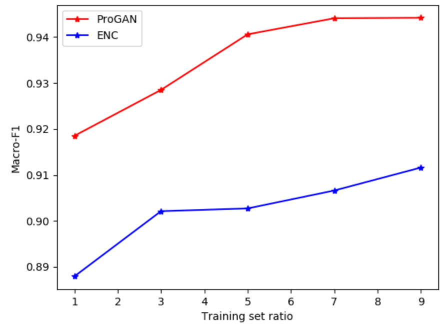
4.2 Experiment Settings

To verify the performance of ProGAN, we compare it with 9 state-of-the-art methods, which include 5 pure network embedding methods: DeepWalk [19], Node2Vec [9], LINE [21], GraRep [2], GraphGAN [23], and 4 attributed network embedding methods: TADW [24], GAE [14], SAGE [10], and DANE [5]. For DeepWalk and Node2Vec, when conducting random walk, we set the number of walks to 10 and the walk length to 80. When constructing the context from node sequences, the window size is set to 10. For LINE, we utilize both the first-order and second-order proximity. In addition, the number of negative samples is 5. For GraRep, the transition step length is set to 5. Throughout our experiments, the dimension of node representations is set to 100.

Regarding our proposed method, when constructing the real triplet, we need to sample negative neighbors. In our experiment, following [21], the sampling probability is $P_v = d_v^{3/4}$ where d_v denotes the degree of node v. When sampling positive neighbors, we randomly select them from the connected nodes of a reference node. In addition, all three components of our proposed ProGAN employ the multi-layer perceptron (MLP) model. Specifically, the architecture of the generator is $128 \longrightarrow 512 \longrightarrow d$ where 128 is the dimension of input noise while d is the dimension of the node attribute. Furthermore, we enforce z_1 and z_2 have the same dimension. Thus, their dimension is 64. The architecture of the encoder is $d \longrightarrow 512 \longrightarrow 100$ where 100 is the dimension of the node representation. The discriminator has an almost same architecture as the encoder. The only difference is that the discriminator has one more branch to distinguish real nodes and generated nodes. Moreover, other than the last layer, the generator uses ReLU as the activation function. Sigmoid function is used in the last layer. For the discriminator and the encoder, LeakyReLU is used as the activation function. In our experiments, to evaluate the learned node representation, we conduct node classification, node clustering, and node visualization on the learned low-dimensional representation. To measure the performance of these tasks, we employ Micro-F1 and Macro-F1 to measure the classification performance while utilizing clustering accuracy (ACC) and normalized mutual information (NMI) to measure the clustering performance. The larger these metrics are, the better the performance is.



(a) Blogcatalog: Micro-F1



(b) Blogcatalog: Macro-F1

Figure 2: The node classification result of ProGAN and ENC. ENC denotes the model without generator and discriminator.

4.3 Results and Analysis

4.3.1 Node Classification. To verify whether a network embedding method can discover and preserve the proximity, we conduct node classification on the learned node representation. In this experiment, the classifier is the ℓ_2-norm regularized logistic regression. In detail, all nodes are used to train network embedding methods. Then, we can get the low-dimensional representation for each node. After that, to conduct node classification, we randomly select 10%, 30%, 50% nodes as the training set and the remained nodes as the testing set. For each case of the training set, we employ five-fold cross-validation to select the best classifier parameters. Then, we report the classification accuracy of the testing set in Table 2, 3, 4, 5. From these tables, we can find that our proposed ProGAN has a better performance than all the other state-of-the-art methods. In particular, we have several observations as follows.

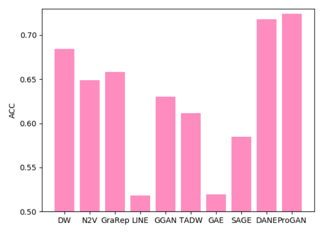
First, comparing with GraphGAN which learns to choose negative neighbors rather than discover the proximity, our method has significant improvement. Especially, ProGAN only uses the popularity-based negative sampling method, but it can improve the embedding result significantly. Thus, we can conclude that the learned proximity is helpful to learn node representations.

Table 2: Node classification result of Citeseer dataset.

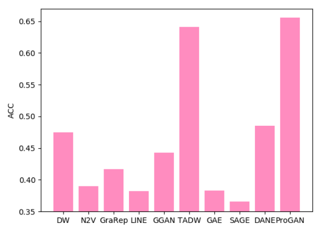
Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.5146	0.4604	0.5623	0.5149	0.5830	0.5397
Node2Vec	0.5059	0.4541	0.5744	0.5249	0.5812	0.5339
LINE	0.4951	0.4472	0.5317	0.4778	0.5395	0.4942
GraRep	0.4908	0.4355	0.5326	0.4622	0.5335	0.4662
GraphGAN	0.4260	0.3837	0.5347	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**

Table 3: Node classification result of Cora dataset.

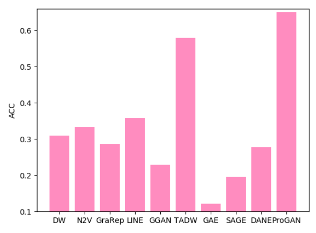
Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.7424	0.7368	0.7975	0.7866	0.8148	0.8032
Node2Vec	0.7777	0.7649	0.8107	0.8001	0.8118	0.8007
LINE	0.7473	0.7399	0.7943	0.7883	0.8081	0.8011
GraRep	0.7609	0.7510	0.7700	0.7558	0.7764	0.7617
GraphGAN	0.6957	0.6804	0.7405	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**



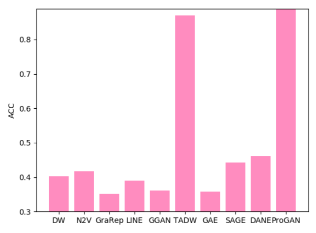
(a) Cora



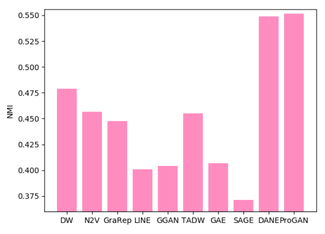
(b) Citeseer



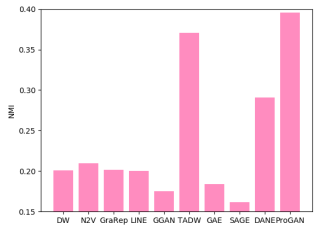
(c) Flickr



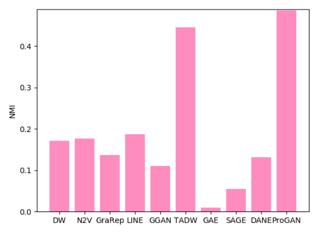
(d) Blogcatalog



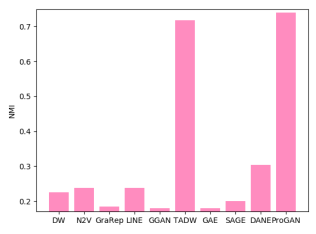
(e) Cora



(f) Citeseer



(g) Flickr



(h) Blogcatalog

Figure 3: Node clustering results. The first row shows ACC while the second row demonstrates NMI.

Table 4: Node classification result of Flickr dataset.

Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.4421	0.4370	0.5022	0.4960	0.5187	0.5108
Node2Vec	0.4682	0.4623	0.5069	0.5029	0.5261	0.5204
LINE	0.5122	0.5013	0.5265	0.5185	0.5399	0.5320
GraRep	0.4759	0.4683	0.5182	0.5106	0.5132	0.5030
GraphGAN	0.4503	0.4466	0.4982	0.4928	0.5206	0.5133
TADW	0.7524	0.7471	0.7707	0.7662	0.7777	0.7738
GAE	0.6273	0.5806	0.6727	0.6055	0.6868	0.6059
SAGE	0.2796	0.2741	0.3194	0.3132	0.3303	0.3253
DANE	0.6078	0.6085	0.6753	0.6747	0.7030	0.7020
ProGAN	**0.7958**	**0.7899**	**0.8192**	**0.8143**	**0.8271**	**0.8239**

Table 5: Node classification result of Blogcatalog dataset.

Method	10%		30%		50%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.6241	0.6156	0.6778	0.6730	0.6999	0.6954
Node2Vec	0.6104	0.6040	0.6501	0.6431	0.6506	0.6455
LINE	0.6630	0.6539	0.6844	0.6737	0.6933	0.6863
GraRep	0.7000	0.6956	0.7163	0.7118	0.7426	0.7398
GraphGAN	0.5196	0.5158	0.5877	0.5811	0.6056	0.6000
TADW	0.9128	0.9115	0.9241	0.9233	0.9296	0.9290
GAE	0.3917	0.3318	0.4219	0.3717	0.4179	0.3681
SAGE	0.5390	0.5321	0.5547	0.5471	0.5387	0.5340
DANE	0.8642	0.8619	0.9065	0.9054	0.9104	0.9091
ProGAN	**0.9294**	**0.9280**	**0.9426**	**0.9418**	**0.9450**	**0.9443**

Second, comparing with attributed network embedding methods which use the high-order proximity, such as GAE and SAGE, our proposed ProGAN shows a better performance. Specifically, when constructing the real triplet, our method only uses the first-order proximity. However, ProGAN can beat those methods which use the high-order proximity. Hence, we can conclude that the generated proximity can serve as the high-order proximity to learn a good node representation.

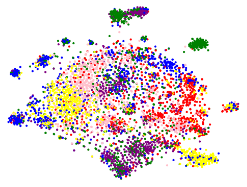
To further verify this point, we conduct another experiment. In detail, we remove the discriminator and generator, only training the encoder with real triplets. Then, we conduct node classification on the learned node representation. The result is shown in Figure 2. Here, this baseline method is denoted by ENC. Due to the space limitation, we only report the results of Blogcatalog. From Figure 2, we can find the improvement of our proposed ProGAN is significant, which verifies that the generated proximity by ProGAN acts as the high-order proximity to benefit node representation learning.

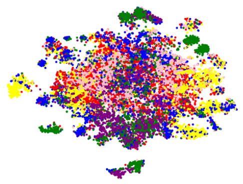
4.3.2 Node Clustering. To further verify the performance of our proposed ProGAN, we conduct node clustering on the learned node representation. Similar with node classification, all nodes are used to train network embedding models. After that, we run K-means on the learned low-dimensional representations. To measure the clustering performance, clustering accuracy (ACC) and normalized mutual information (NMI) are used. The result is shown in Figure 3. From it, we can find that our proposed ProGAN can outperform the other baseline methods significantly in terms of both ACC and NMI, which further verifies the effectiveness of our proposed method.

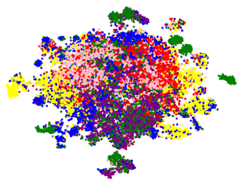
4.3.3 Network Visualization. Network visualization is another widely used task to verify the embedding result. Hence, we visualize the learned node representations of these methods by using T-SNE [16]. Specifically, it projects the learned low-dimensional nodes into the two-dimensional space and then visualizes them. Due to the space limitation, we only report the result of Blogcatalog dataset. The result is shown in Figure 4. Here, we only report the result of the best 7 baseline methods. Apparently, comparing with the other state-of-the-art methods, the learned node representation of ProGAN has a more compact structure within a group and a larger margin between different groups. In particular, although TADW and DANE have good node classification result, yet its node representations do not have a compact group structure and large margin between groups as our method, which is also verified by the clustering result. All in all, ProGAN can discover and preserve the proximity better than the baseline methods.

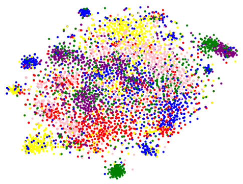
5 CONCLUSION

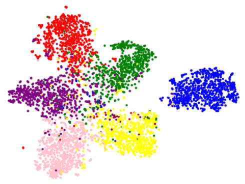
In this paper, we propose a novel proximity generative adversarial network for network embedding. Specifically, the proposed ProGAN can generate proximities, with which we can discover the

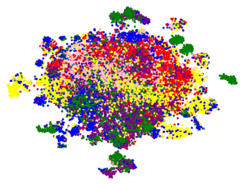

(a) DeepWalk

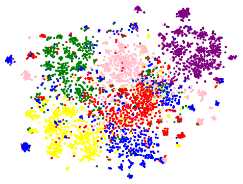

(b) GraRep


(c) LINE


(d) GraphGAN


(e) TADW


(f) SAGE


(g) DANE

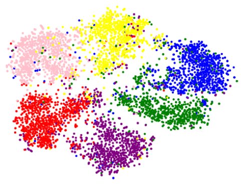

(h) ProGAN

Figure 4: Network visualization of Blogcatalog dataset.

underlying complicated relationship between different nodes. Then, by preserving these underlying proximities in the low-dimensional space, the learned node representation has shown better performance on several tasks, which verifies the effectiveness of our proposed method. For the future work, we will focus on generating the structural proximities rather than the ordinary proximity to further improve network embedding.

REFERENCES

[1] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815* (2017).

[2] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 891–900.

[3] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 119–128.

[4] Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. 2017. Adversarial network embedding. *arXiv preprint arXiv:1711.07838* (2017).

[5] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding.. In *IJCAI*.

[6] Hongchang Gao and Heng Huang. 2018. Self-paced network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1406–1415.

[7] Hongchang Gao, Jian Pei, and Heng Huang. 2019. Conditional Random Field Enhanced Graph Convolutional Neural Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.

[8] 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.

[9] 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.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[11] 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.

[12] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 731–739.

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

[14] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[16] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[17] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.

[18] S Pan, J Wu, X Zhu, C Zhang, and Y Wang. 2016. Tri-party deep network representation. In *International Joint Conference on Artificial Intelligence*. AAAI Press/International Joint Conferences on Artificial Intelligence.

[19] 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.

[20] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 385–394.

[21] 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.

[22] 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.

[23] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2017. Graphgan: Graph representation learning with generative adversarial nets. *arXiv preprint arXiv:1711.08267* (2017).

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

[25] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2016. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *arXiv preprint arXiv:1609.05473* (2016).

[26] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *IEEE International Conference on Computer Vision*.