Representation Learning on Graphs by Integrating Content and Structure Information

Ayush Maheshwari*, Ayush Goyal*, Amit Kumar†, Manjesh Kumar Hanawal*, Ganesh Ramakrishnan*
*Indian Institute of Technology Bombay
Mumbai, India
{ayushm, ayushgoyal, ganesh}@cse.iitb.ac.in,
mhanawal@iitb.ac.in
†Adobe Systems
Bengaluru, India
scienceamit1@gmail.com

***Abstract*—The problem of representation learning on graph can be difficult due to limited knowledge of training data and large presence of missing edges. Real-world social networks do not provide complete information about the network due to hidden information and privacy constraints. In such scenarios, typical representation learning methods are not able to capture network information effectively. In order to make them more useful, any available feature information can be used in addition to the network structure. In this paper, we aim to learn better representations by exploiting both content (or feature) information of nodes and structural information of the network. Our approach leverages generative adversarial networks to learn embedding for generator and discriminator in a minimax game. While the generator estimates the neighborhood of a node, the discriminator distinguishes between the presence or absence of a link for a pair of nodes. We demonstrate the effectiveness of our approach on five real-world publicly available datasets on the problems of link prediction and node classification. On both tasks, we achieve significant gains, outperforming current state-of-the-art methods by considerable margins. Our code is available on Github[1].**

***Index Terms*—graph representation learning, node embedding, link prediction, node classification**

I. Introduction

Graphs are a ubiquitous data structure representing relational data from domains such as social networks, biological networks, knowledge graphs, world wide web networks, language networks, *etc.* Graphs can be used to analyse the interconnection structure among the data. It has resulted in tremendous advancements in tasks such as link prediction, community discovery, node classification, *etc.* For instance, in social networks analysis of connections between the users reveal about their preferences and interactions.

Representation learning on graphs has received considerable attention in recent years in many fields such as social networks, protein-protein interaction networks, recommendation engines, *etc.* *Graph Representation Learning* aims to represent higher dimension network structure into a lower dimensional embedding. It represents vertex/node into a low-dimensional space that can be used for prediction, classification and clustering tasks. In many applications, in addition to the network structure feature information about each node might be available and can provide complementary information about the network, especially in cases where full information about the network structure is not available. In this paper, we study graph representation methods that effectively use any feature information available for generating better node embeddings.

Traditional approaches for graph representation use matrix factorization based methods [1]–[3] to reduce the dimensionality of the embeddings. These approaches construct a similarity graph for the nodes such that distance between two connected nodes is minimum. However, these approaches suffer from scalability issues having time complexity of $O(|E|)d^2$ where E is the set of edges in the graph and d is the number of dimensions. LINE (Large-scale Information Network Embedding) [4] preserves first-order and second-order proximity in the graph. It considers immediate and shared neighbors of each node to capture local and global information in the network. Recent network representation techniques [5]–[8] use first order proximity to approximate node embedding. They utilize link structure (or neighborhood information) to represent a node in a lower dimensional vector space. These methods fail to perform when the network is small or only a sub-graph is given [9]. Even when the training data contains large number of missing edges, these methods do not perform as expected. Deepwalk [7] uses short random walks to sample vertices. It adopts neural language modeling technique Skip-Gram [10] to maximize the likelihood of observing context vertices for a given vertex. Node2vec [5] extends Deepwalk by introducing biased random walk procedure providing flexibility while sampling context vertices.

In real-world networks such as social networks, complete information of a node is generally not available due to user and privacy constraints. For example, in social networks users tend to hide their information such as friendship networks, user likes, *etc.* Therefore, to gain a proper understanding of the network it is crucial to consider any available additional information. Moreover, within a limited data availability setting such information play a critical role in generating effective embeddings. There are a few attempts to generate embeddings by integrating content [11]–[13]. Yang [11]

[1]https://github.com/ayushbits/node-embeddings

978-1-5386-7902-9/19/$31.00 ©2019 IEEE

extends Deepwalk [7] by including text information as a feature matrix. It factorizes adjacency matrix M into the product of three matrices and concatenates text information matrix to form embeddings. GraphSAGE [13] leverage node features to learn an embedding function that can generalize on unseen nodes. It learns *aggregator functions* that aggregates feature information from neighborhood nodes and evaluated results on node classification.

In this paper, we leverage vertex neighborhood and content information to learn node embedding that outperforms all previous link-prediction benchmarks. We leverage Generative Adversarial Nets (GAN) that unifies the generative and discriminative approach for graph representation learning. We train generative and discriminative models during the learning of GAN. 1) Generator $G(v|v_c, f)$ tries to maximize the underlying true distribution, 2) Discriminator $D(v, v_c)$ tries to distinguish between actual pairs and samples generated by generator pairs. In this process, generator learns to identify neighborhood of vertex v and discriminator learns to differentiate between connected and non-connected pairs. In this work, we use node features along with structural information in the generator and discriminator model. Then, generator and discriminator act as two players in the *minimax* game improving their embeddings. The process is repeated until the generator is indistinguishable from the true connectivity distribution.

Inspired by GAN, GraphGAN [8] exploits structure information to learn graph representation. It outperformed all previous approaches on link prediction tasks. However, it performs poorly on node classification task on sparse networks. We demonstrate their results for node classification in Section IV-B. There has been a lack of consistency in evaluation methods adopted by previous approaches. GraphGAN [8] evaluates on equal samples of positive and negative edges. In reality, positive samples are way less than negative samples. Evaluation metrics are discussed in detail in Section IV-A.

We introduce content information in addition to the structural information in both generator and discriminator. We propose a random-walk based sampling strategy for the generator, accounting for both structural and content information. We achieve more than 20% gains over current baselines on link prediction. On node classification, we obtain 20% improvement over state-of-the-art baselines. Again, using the Wilcoxon signed-rank test, we notice that the sum of the signed ranks is very clearly in favour of our approach as against Node2Vec and GraphGAN.

Contributions : Our contributions are as follows:

- We motivate the use of content along with structure information by evaluating results of previously proposed approaches on link prediction and node classification.
- We propose a generator and discriminator based approach to learn embeddings by incorporating features.
- We evaluate the performance of our proposed algorithms on benchmark datasets with different training and testing sizes for link prediction and node classification. The results are compared with various state-of-the-art embedding algorithms to show the superiority of our approach. We also discuss different evaluation methods adopted in previous approaches and motivate the use of current evaluation metrics.

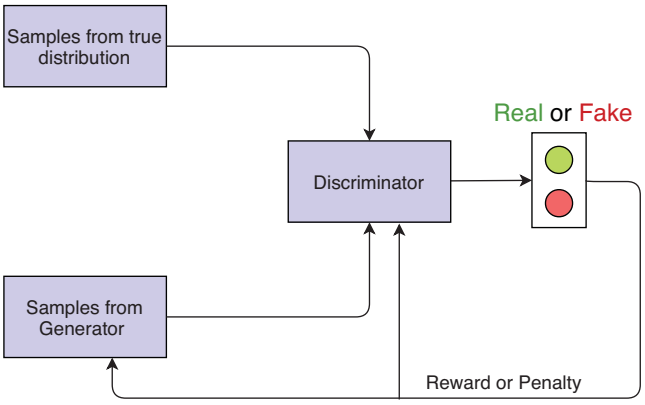

Fig. 1: Illustration of GAN framework

II. Problem Formulation

Let $\mathscr{G} = (V, E)$ be a given undirected graph, where V is the vertex set $V = v_1, v_2, \ldots, v_n$ and E is the associated edge set $E = (v_i, v_j)$ such that $v_i, v_j \in V$. For a given vertex v_c, we define neighborhood as $N(v_c)$ which are the set of vertices directly connected to v_c. We define content information as f_c which is the feature set for node v_c. f_c is a vector containing binary values[2] from the feature set F. We denote the underlying true connectivity distribution for a vertex v_c as a conditional probability $p_{true}(v|v_c, f_c)$. It specifies that v is conditionally dependent on all other vertices in V and their respective features f_c. In our setting, generator tries to approximate the true distribution $p_{true}(v|v_c, f_c)$ whereas discriminator learns to differentiate between pair of vertices (v_i, v_j). Generative Adversarial Networks (GAN) [14] is an adversarial modeling framework that estimates generative models. We simultaneously train both generator G and discriminator D in an adversarial manner expecting G to learn the data distribution. G maximizes the probability of deceiving D whereas D maximizes its probability of differentiating between generator samples and true samples. Reward or penalty mechanism ensures that D's output are feed back to G to guarantee joint training. GAN framework is illustrated in Figure 1.

Generator: It captures the data distribution and learns a parameter θ_g such that $G(v|v_c, f_c; \theta_g)$ can approximate the true distribution. It tries to generate vertices that are most likely to be connected to vertex v_c from vertex set V. In addition to neighborhood information, generator considers feature matrix f_c for the vertex v_c.

[2]Continuous valued feature can also be handled using an appropriate distance metric

Discriminator: It estimates a probability that the sample came from the the training data. It learns a parameter θ_d such that $D(v_i, v_j; \theta_d)$ can discriminate between the presence or absence of an edge. It outputs a score representing the probability of existence of an edge between v_i and v_j.

The minimax game with objective function $V(G, D)$ can be formalised as follows:

$$\min_{\theta_G} \max_{\theta_D} V(G, D) = \sum_{c=1}^{V} \Big(\mathbb{E}_{v \sim p_{true}(\cdot|v_c)} [\log D(v, v_c; \theta_D, w_D)] + \mathbb{E}_{v \sim G(\cdot|v_c, f_{v_c}; \theta_G, w_G)} [\log (1 - D(v, v_c; \theta_D, w_D))] \Big) \quad (1)$$

Eq. (1) explains our objective function wherein generator and discriminator learns by alternately maximizing and minimizing the objective function. Discriminator D is trained with equal number of positive samples from $p_{true}(\cdot|v_c)$ and negative samples from generator $G(\cdot|v_c; \theta_g, w_G)$. Discriminator D learns to discriminate between positive and negative samples and feedback its gradient to the generator G. Generator G receives rewards/penalty from the discriminator D and therefore learns to imitate the true distribution. We run generator until G is indistinguishable from the true distribution.

A. Discriminator

Objective function of discriminator D maximizes log probability of assigning correct labels to positive samples from true distribution and negative samples from generator. We define D as a sigmoid function of the inner product of two input vertices and element-wise $XNOR$ of two input feature vectors[3].

$$\begin{aligned} D(v, v_c) =& \sigma(d_v^T, d_{v_c}, f_v, f_{v_c}, w) \\ =& \frac{1}{1 + exp(-d_v^T d_{v_c} + [w \circ w](f_v \odot f_{v_c}))} \end{aligned} \quad (2)$$

where $d_v, d_{v_c} \in \mathbb{R}^m$ are the *m*-dimensional representation vectors of vertices v and v_c, respectively, and $f_v, f_{v_c} \in F$ are their associated feature vectors. w is the weight vector and $f_v \odot f_{v_c}$ denotes the element-wise $XNOR$ between f_v and f_{v_c}. $w \circ w$ denotes a Hadamard product [15] of weight vectors w. It refers to element-wise multiplication of two vectors. The union of all d_vs represents the parameter θ_D of the discriminator.

We perform element-wise $XNOR$ of feature vectors in the discriminator optimization while retaining structural properties of network. Gradient computation can be solved by stochastic gradient ascent if D is differentiable with respect to θ_D. Gradient ascent update equation with respect to d_v and d_{v_c} can be written as :

$$\nabla_{\theta_D} V(G, D) = \begin{cases} \nabla_{\theta_D} \ log \ D(v, v_c), \ if \ v \sim p_{true}; \\ \nabla_{\theta_D} \Big(1 - log \ D(v, v_c)\Big), \ if \ v \sim G \end{cases} \quad (3)$$

[3]When the features are continuous, element-wise absolute distance can be used

Gradient ascent update equation for w can be written as :

$$\nabla_{w_D} V(G, D) = \begin{cases} \nabla_{w_D} \ logD(v, v_c), if v \sim p_{true}; \\ \nabla_{w_D} \ \Big(1 - log \ D(v, v_c)\Big), \ if \ v \sim G \end{cases} \quad (4)$$

B. Generator

Generator G tries to approximate the true connectivity distribution and aims to minimize the log-probability that discriminator correctly assigns negative labels to the samples generated by G. We define generator as a softmax function over all the vertices in V and element-wise $XNOR$ of two input feature vectors. The equation can be written as:

$$G(v|v_c) = \frac{exp\Big((g_v^T g_{v_c}) + [w \circ w](f_v \odot f_{v_c})\Big)}{\sum_{v \neq v_c} exp\Big((g_v^T g_{v_c}) + [w \circ w](f_v \odot f_{v_c})\Big)} \quad (5)$$

where $g_v, g_{v_c} \in \mathbb{R}^m$ are the m-dimensional representation vectors of vertex v and v_c for generator G, $f_v, f_{v_c} \in F$ are the feature vectors for the vertices v and v_c respectively. The union of all g_vs represents the parameter θ_G of the generator. The calculation of softmax in the equations considers all vertices in V thereby calculating gradient θ_G for all vertices at each step. Gradient computation may become inefficient for large networks wherein a large number of nodes exist. In graph networks, immediate vertices or neighboring vertices are of interest for retaining structural properties. Moreover, this function does not assign higher weights to neighboring connections but assigns equal weights to all nodes in the graph. It is computationally inefficient to calculate softmax over all the vertices in the graph. As in [8], we may modify Eq (5) to compute gradient over neighboring vertices only. GraphGAN [8] provides detailed explanation of choosing *normalized, graph-structure-aware, and computationally efficient* softmax function.

$$p(v_i|v) = \frac{exp\Big((g_{v_i}^T g_v) + [w \circ w](f_{v_i} \odot f_v)\Big)}{\sum_{v_j \in N_c(v)} exp\Big((g_{v_j}^T g_v) + [w \circ w](f_{v_j} \odot f_v)\Big)} \quad (6)$$

Eq (6) is a softmax function over all neighboring vertices of v. We can compute the gradient of $V(G, D)$ with respect to θ_G as :

$$\nabla_{\theta_G} V(G, D) = \sum_{c=1}^{V} \mathbb{E}_{v \sim G(\cdot|v_c)} \Big[\nabla_{\theta_G} logG(v|v_c) \ log\Big(1 - D(v, v_c)\Big) \Big] \quad (7)$$

Similarly, we can compute the gradient with respect to w as :

$$\nabla_{w_G} V(G, D) = \sum_{c=1}^{V} \mathbb{E}_{v \sim G(\cdot|v_c)} \Big[\nabla_{w_G} logG(v|v_c) \ log\Big(1 - D(v, v_c)\Big) \Big] \quad (8)$$

The above equation suggests that vertices with larger probability of negative samples will be penalised by the generator

due to feedback from discriminator. Inspired by GraphGAN [8], pseudo-code of our approach is described in Algorithm 1.

Algorithm 1 Proposed approach for generating embeddings

Require: Number of vertices to sample from G and D
Ensure: Embedding matrix for G and D
1: Initialize G and D with pre-trained or random initializations;
2: Construct BFS tree B_v for all $v_c \in V$;
3: **while** G does not converge **do**
4: **for** G-steps **do**
5: Generate g vertices for each vertex v_c using Algorithm 2;
6: Update θ_G and w_G according to Eq. (6), (7) and (8);
7: **end for**
8: **for** D-steps **do**
9: Sample t positive vertices from ground truth and t negative vertices from G for each vertex v_c;
10: Update θ_D and w_D according to Eq. (2), (3) and (4);
11: **end for**
12: **end while**

C. Sampling strategy for generator

Previous approaches, such as Deepwalk [7] and LINE [4] adopt random walk to choose sampling vertices. However, such an unbiased random walk has an equal probability of choosing samples vertices from the graph $\mathscr{G}$. Node2vec [5] manipulate hyperparameters p and q to adjust the frequency of choosing sample vertices in a biased random walk. In this paper, we perform a weighted random walk on all the vertices in graph $\mathscr{G}$ starting at vertex v_c. For a given vertex v_c, probability of choosing vertex v_i is calculated from Eq (6). Probability score considers both structural and content information while performing the random walk. It suggests that vertex can be sampled even if a pair of vertices are not directly connected but have a common set of features.

Firstly, we create Breadth First Search (BFS) tree B_v for the vertex set V. For the starting vertex v_c, we move to the next adjacent vertex with probability estimated from Eq (6). The process is repeated until the previous vertex is chosen for sampling. Starting vertex v_c and the current vertex is chosen as sample pairs. For a given vertex v_c, the algorithm stops in $O(logV)$ steps due to the tree structure of vertices. Algorithm 2 describes the sampling procedure.

We discuss time complexity for the proposed approach. Construction of BFS tree in line 2 requires $O(V(V+E)) = O(dV^2)$ since tree is constructed for all the vertices in the graph and each BFS tree has time complexity of $O(V + E)$ [16]. Each iteration of Generator (line 5 and 6) and Discriminator (line 9 and 10), has time complexity of $O(V\ logV)$. The addition of features will slightly increase the computation time but it will be negligible compared to the size of V.

III. Experiments

In this section, we evaluate the performance of our proposed algorithm on real-world datasets. We compare our results with state-of-the-art network embedding algorithms. We choose four real-world datasets which contain both structure and content information. To evaluate the effectiveness of the generated embeddings, we evaluate embedding on the task of link prediction and node classification. We find that our model outperforms all previous state-of-the-art embedding algorithms. Table I provides summary of the datasets.

Algorithm 2 Sampling strategy for generator

Require: BFS tree B_v representation of $v \in V$
Ensure: Generated samples v_{gen}
1: Select a starting vertex $v_c := v_{cur}$ and $v_c := v_{prev}$;
2: **while** true **do**
3: Select neighboring vertex v_i proportional to $p(v_i|v_c)$ using Eq. (6);
4: **if** $v_i = v_{prev}$ **then**
5: RETURN v_{gen};
6: **else**
7: $v_{prev} := v_{cur}, v_{cur} := v_i$
8: **end if**
9: **end while**

TABLE I: Summary of datasets used for the experiments.

Dataset	#Nodes	#Edges	#Features	#Classes
Facebook	4038	88234	1283	-
Google Plus	1650	166292	1319	-
Twitter	246	9630	2263	-
Citeseer	3312	4715	3703	6
Cora	2708	5429	1433	7

A. Datasets

We used following four real-world datasets for our experiments :

- **Facebook**[4]: Facebook dataset contains 4039 nodes, 88234 edges and 1283 features. The features are binary values representing the presence or absence of the features. For instance, Gender has two features, namely, Female and Male that are represented as 1 and 0 in the feature set.
- **Google Plus**[5]: The original Google plus dataset contains 107614 nodes and 13673453 edges. We choose an egonet (subgraph) from the dataset containing 1650 nodes and 166292 edges. The dataset contains 1319 features organized in a similar manner as Facebook dataset.
- **Twitter**[6]: Similar to Google plus dataset, we choose an egonet from the large twitter dataset. The evaluated dataset contains 246 nodes and 9630 edges with 2263 features consisting of unique words.
- **Citeseer**[7]: It is a citation network consisting of scientific publications. Node represents publications and edges

[4]http://snap.stanford.edu/data/ego-Facebook.html
[5]http://snap.stanford.edu/data/ego-Gplus.html
[6]http://snap.stanford.edu/data/ego-Twitter.html
[7]https://linqs.soe.ucsc.edu/data

signifies citations with other publications in the network. Network contains 3312 nodes and 413 edges. It has 6 classes and 3703 features consisting of unique words.

- **Cora**[8]: It is a citation network having 2708 scientific publications classified into one of seven classes. It contains 5429 links and features consists of 1433 unique words.

B. Experiment Setup

Datasets are chosen such that dense and sparse networks exist with a large number of features. In citeseer network, the number of edges is less than the number of nodes and a large number of features are present. Sparse networks can evaluate the performance of our feature-based approach. The dense network such as Google plus can evaluate the effectiveness of our algorithm.

We compare our proposed approach with GraphGAN [8] and Node2vec [5]. GraphGAN experiments on link prediction have outperformed previous approaches such as DeepWalk [7], LINE [4], Struc2vec [6] and node2vec. Performance of DeepWalk and struc2vec performs relatively poor in link prediction due to their limitation to explore the local neighborhood of nodes. Node2vec is a variant of Deepwalk and capture link prediction tasks more accurately due to the inclusion of biased random walks. All these methods including GraphGAN incorporate structural information to learn node embeddings. We exclude other approaches such as DeepWalk, LINE, struc2vec that are already shown inferior to GraphGAN.

C. Parameter Settings

We perform stochastic gradient descent to update parameters with a learning rate of 0.0001 on batch size of 64 for both generator and discriminator. In each iteration, we set 20 as the number of positive and negative samples, and run 30 steps of generator and discriminator. The system is trained until the generator is indistinguishable from the true distribution. We have used $L2$ regularization term both generator and discriminator. Dimension size of output embeddings is 32. We choose the above parameters based on cross-validation. For experiments on GraphGAN and node2vec, best hyper-parameters are chosen for training the models.

IV. Results

In this section, we present the results of our experiments for link prediction and node classification.

A. Link Prediction

The aim of link prediction is to predict whether there exists an edge between a pair of vertices or not. This task shows the performance of our embeddings for the problem of edge prediction. For this task, we test on two scenarios of training set size. Firstly, we randomly select 20% positive edges[9] for training and remaining 80% positive edges for testing. Secondly, we randomly select 40% positive edges for training and the remaining 60% positive edges for testing. For each pair of vertices in the testing set, we calculate the score by adding dot product of the embeddings and weighted features. Then, we use a sigmoid activation function to keep values between 0 and 1. The threshold of activation function regulates precision values for positive and negative edges. Moreover, the threshold may be different for various embedding methods but we report precision numbers for the optimum threshold. The results for link prediction on five datasets are shown in Table II.

We randomly choose 5x negative edges for evaluation purpose, i.e., number of negative edges = $5\times$ positive edges. In real-world networks, the absence of links is way higher than the presence of links. It is important that we consider this imbalance in the testing set. In previous evaluation approaches such as GraphGAN [8], the equal number of positive and negative edges are given in the testing set which does not represent real-world networks. In this paper, we adopt the test set ratio of negative to positive edges as $5:1$.

In Table II, P@0 and P@1 refer to precision for negative and positive edges respectively. It is calculated as a fraction of edges predicted correctly. Due to our imbalanced positive and negative edges in the testing set, we calculate the weighted macro F1 score as shown in Eq (9) and (10).

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{9}$$

$$Weighted\ Macro\ F1 = \frac{5 \times P@0 + P@1}{5+1} \tag{10}$$

We observe that the performance of our approach outperforms all baselines in link prediction. Addition of features (or content) information boosts link prediction. For facebook dataset, in case of 20% training set, node2vec is able to beat link prediction when the link is present, however it poorly predicts when the link is absent. Overalll, our approach report better Macro-F1 scores than other approaches. Further, we observe that node2vec is biased towards predicting positive edges correctly and predict relatively poor on the absence of links on Facebook, Citeseer and Cora dataset. The training process of node2vec that takes structural properties into consideration explains such imbalance. For Facebook dataset on 20% training set, our numbers are slightly less than GraphGAN but on all others train-test ratio, we outperform both embedding methods. Our model outperforms other methods by a considerable margin of more than 20% where there is a large number of missing edges such as Citeseer and Cora datasets. Our approach of incorporating content information in addition to structure information in a adversarial training process during training produces improved embedding. Using the Wilcoxon signed-rank test, we notice that the sum of the signed ranks is very clearly in favour of our approach as against Node2Vec and GraphGAN.

[8]https://relational.fit.cvut.cz/dataset/CORA

[9]Positive edge refers to the presence of an edge and negative edge refers to the absence of an edge between a pair of vertices.

TABLE II: Performance of the algorithms for link prediction on various datasets for different embedding methods on Facebook, Google Plus, Twitter, Citeseer and Cora datasets. 20/80 and 40/60 represents train-test ratio. Algorithms were trained with 20% and 40% positive edges and test on remaining 80% and 60% positive edges respectively. P@0 and P@1 represents precision for negative edges and positive edges respectively. Macro-F1 represents weighted Macro-F1 scores for positive and negative edges with a ratio of 1:5.

(a) Facebook Dataset

Algorithm	20/80			40/60		
	P@0	P@1	Macro-F1	P@0	P@1	Macro-F1
Node2Vec	0.36	**0.97**	0.45	0.75	0.98	0.79
GraphGAN	0.5	0.5	0.5	**0.92**	**0.72**	**0.89**
Our Approach	**0.54**	0.54	**0.54**	0.91	0.71	0.88

(b) Google Plus Dataset

Algorithm	20/80			40/60		
	P@0	P@1	Macro-F1	P@0	P@1	Macro-F1
Node2Vec	0.62	0.69	0.63	0.44	0.88	0.52
GraphGAN	0.67	0.71	0.68	0.52	0.91	0.584
Our Approach	**0.68**	**0.72**	**0.69**	**0.52**	**0.91**	**0.586**

(c) Twitter Dataset

Algorithm	20/80			40/60		
	P@0	P@1	Macro-F1	P@0	P@1	Macro-F1
Node2Vec	0.60	0.66	0.61	0.64	0.67	0.65
GraphGAN	0.64	0.7	0.65	0.65	0.67	0.65
Our Approach	**0.66**	**0.7**	**0.664**	**0.65**	**0.69**	**0.66**

(d) Citeseer Dataset

Algorithm	20/80			40/60		
	P@0	P@1	Macro-F1	P@0	P@1	Macro-F1
Node2Vec	0.46	**0.90**	0.53	0.47	0.75	0.52
GraphGAN	0.50	0.51	0.50	0.50	0.52	0.50
Our Approach	**0.68**	0.70	**0.69**	**0.69**	**0.78**	**0.70**

(e) Cora Dataset

Algorithm	20/80			40/60		
	P@0	P@1	Macro-F1	P@0	P@1	Macro-F1
Node2Vec	0.53	0.70	0.55	0.43	0.56	0.45
GraphGAN	0.49	0.51	0.50	0.48	0.56	0.49
Our Approach	**0.59**	**0.70**	**0.60**	**0.56**	**0.72**	**0.58**

B. Node Classification

Node classification is an important application in cases where we need to predict labels of a vertex. Each vertex is assigned one or more label, our aim is to predict label of a vertex. The performance of node classification can reveal the distinguishing ability of various embedding methods. In this task, we retrieve the embeddings of the nodes and use them as features to train a logistic regression [17]. We perform this experiment for the different train-test ratio. The dataset is split for training set size of 5%, 50% and 90% and used remaining nodes for testing. We take two popular metrics based on the F1 score as evaluation criteria, namely, Macro-F1 and Micro-F1. In Macro-F1, we average the performance of each individual class whereas in Micro-F1 we calculate the individual performance of each class on true positives, false positive, false negative and true negative. Generally, the higher the score, better the classification performance. We repeat the same experiment for other embedding methods.

The results for node classification for Citeseer and Cora datasets are presented in Table III. We observe that the performance of Node2vec is higher than GraphGAN for both datasets and training set size. On both metrics, our model outperforms previous approaches by a significant margin. For citeseer dataset, we observe a jump of 10% - 20% of both Macro-F1 and Micro-F1 scores. It turns out that our approach outperforms baselines by a higher margin for node classification. This indicates that features play a crucial role to encode node representations for link prediction and node classification.

TABLE III: Performance of the algorithms for node classification for various embedding methods on the Citeseer and Cora datasets. We have tried methods on different train-test ratio namely 5%-95%, 50%-50% and 90%-10%. Macro-F1 and Micro-F1 results are shown for the embedding methods.

(a) Citeseer Dataset

Metric	Algorithm	5/95	50/50	90/10
Macro-F1	Node2Vec	0.45	0.55	0.56
	GraphGAN	0.34	0.40	0.40
	Our Approach	**0.54**	**0.65**	**0.68**
Micro-F1	Node2Vec	0.50	0.60	0.62
	GraphGAN	0.38	0.46	0.40
	Our Approach	**0.60**	**0.65**	**0.68**

(b) Cora Dataset

Metric	Algorithm	5/95	50/50	90/10
Macro-F1	Node2Vec	**0.69**	0.74	0.81
	GraphGAN	0.34	0.44	0.49
	Our Approach	0.57	**0.74**	**0.82**
Micro-F1	Node2Vec	**0.69**	0.77	0.8
	GraphGAN	0.39	0.47	0.52
	Our Approach	0.61	**0.77**	**0.84**

C. Significance Test

On all datasets, we report significance using the Wilcoxon signed-rank test [18]. This is a non-parametric test used to

compare median performance of a pair of algorithms. It evaluates null hypothesis that there is no significant difference between a pair of algorithms. In the signed rank test, one first rank the absolute value of the differences observed in the performance of the pair of algorithms and signs are assigned depending on the whether performance of first algorithm is higher than that of the second. If there is no significant difference (that is, if the null hypothesis holds), the sum of the signed ranks should be approximately 0. In our case, we obtain a sum of 134 and 128 for Node2vec and GraphGAN respectively. We note that comparing a pair of algorithms using the Wilcoxon test is equivalent to determining if the area under the ROC curves of the algorithms differ significantly. The null hypothesis is that our model does not perform better than two baselines and the alternative hypothesis is that our model performs better than baselines. We obtain a $p - value \leq 0.01$ clearly reject the null hypothesis and support our claim. Table IV shows the results of Wilcoxon signed-rank test.

TABLE IV: Test for statistical significance of our model with Node2vec and GraphGAN using Wilcoxon Signed Rank test. Sum refers to sum of signed ranks.

Algorithm	Sum	Z-value	W-value	p-value
Node2vec	134	-3.5162	0	0.00044
GraphGAN	128	-3.3611	3	0.00078

V. Conclusion and Future Work

In this paper, we leverage generative adversarial network to train our network. We train generator and discriminator in a minimax game utilizing both structure and feature information to learn node embedding. Similar to GAN, the generator is trained to utilize discriminator output to improve its performance. We conduct extensive experiments on five datasets for link prediction and node classification. We observe significant gains in both tasks outperforming baselines by a significant margin. For node classification, our model achieves 10%-20% higher Macro-F1 and Micro-F1 score over current state-of-the-art baselines. In link prediction, our model registers at least 20% higher Macro-F1 scores over current baselines on the sparse networks. In well-connected networks, our model exceeds current state-of-the-art approaches Macro-F1 scores.

In the future work, we propose to add features directly to the embeddings. We would also like to introduce feature selection to reducing computation time. We propose to extend our idea to time-variant networks such as social networks where network relationship changes with time.

VI. Acknowledgements

We thank Saket Bhojane for his useful contributions in initial experiments. We would also like to thank National Center of Excellence in Technology for Internal Security (NCETIS) at IIT Bombay for financially supporting the first author. Manjesh Kumar Hanawal would like to thank support from INSPIRE faculty fellowships from DST, Government of India and SEED grant (16IRCCSG010) from IIT Bombay.

References

[1] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in neural information processing systems*, 2002, pp. 585–591.

[2] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 891–900.

[3] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 1105–1114.

[4] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[5] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[6] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 385–394.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[8] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "GraphGAN: Graph representation learning with generative adversarial nets," *arXiv preprint arXiv:1711.08267*, 2017.

[9] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 2, p. 10, 2011.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[11] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information." in *IJCAI*, 2015, pp. 2111–2117.

[12] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 2017, pp. 731–739.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[15] R. A. Horn, "The hadamard product," in *Proc. Symp. Appl. Math*, vol. 40, 1990, pp. 87–169.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[17] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[18] S. Siegel and N. Castellan, 2006.