Adversarially Generating Rank-Constrained Graphs

William Shiao
Dept. of Computer Science and Engineering
University of California Riverside
Email: wshia002@ucr.edu

Evangelos E. Papalexakis
Dept. of Computer Science and Engineering
University of California Riverside
Email: epapalex@cs.ucr.edu

***Abstract*—Graph generation is a task that has been explored with a wide variety of methods. Recently, several papers have applied Generative Adversarial Networks (GANs) to this task, but most of these methods result in graphs of full or unknown rank. Many real-world graphs have low rank, which roughly translates to the number of communities in that graph. Furthermore, it has been shown that taking the low rank approximation of a graph can defend against adversarial attacks. This suggests that testing models against graphs of different rank may be useful.**

However, current methods provide no way to control the rank of generated graphs. In this paper, we propose two variants of BRGAN: GAN architectures that generates synthetic graphs, which in addition to having realistic graph features, also have bounded rank. Our first variant, BRGAN-A, generates synthetic graphs competitive with state-of-the-art models, with rank equal to or lower than the desired rank. Our second variant, BRGAN-B, generates graphs of almost exactly the desired rank, but results in less realistic results. We also propose a novel rank penalty term on the generator, which allows us to control this realism-rank tradeoff.

I. Introduction

Graph generation is a common task, and many models exist for this task. Traditional statistical methods (such as the Barabási–Albert model [1]) usually attempt to model specific attributes of the graph. While this can lead to generated graphs being similar to the target graph with respect to one or more attributes, it requires the user to accurately identify characteristics of the graph they wish to mimic. This issue has been addressed with the introduction of neural network models (like GraphRNN [2]), which learn directly from a set of graphs without requiring the user to explicitly declare any graph attributes.

Recent advances in the field include the use of Generative Adversarial Networks (GANs) for graph generation. The GAN was first introduced in 2014 by Goodfellow *et al.*[3]. While they were originally used for image generation, they have since been applied to a wide variety of fields, including the field of graph generation.

A current state-of-the-art model LGGAN [4] learns to directly generate the adjacency matrices of graphs and its corresponding labels given a set of graphs as input. It does this by using a GAN with a Graph Convolutional Network (GCN) [5] as its discriminator and a Multi-layer Perceptron (MLP) network as its generator.

Alternatively, GraphRNN [2] models the graph as a sequence and uses a RNN to generate realistic graphs. You *et al.*[2] also provides a set of datasets that we use to evaluate the quality of our model.

Most existing graph generation techniques tend to produce a graph of full or high rank. While this is sufficient for most use cases, generating a graph of a known rank can also be useful. The rank of a community graph roughly corresponds to the number of communities within the graph. Bounding the rank of the generated graph therefore allows a user to generate graphs with bounded number of communities, which is difficult or impossible with existing graph generation models and can result in more realistic graphs.

Low-rank approximations of graphs have also been shown to be useful in defending against adversarial attacks [6], which further suggests that the rank of a graph is an useful parameter. Being able to control the rank allows users to generate synthetic datasets with realistic graphs to test the effectiveness of new models on inputs with different ranks.

Determining the full rank of a matrix is computationally tractable by using the Singular Value Decomposition (SVD), which can be calculated in polynomial time. However, determining the low-rank subspace in which the data "live" is by no means a trivial problem, and is usually done heuristically by truncating the singular values of a matrix so that a certain percentage of the data variation is preserved.

In matrices, this is typically not considered a major issue, however, the problem is further compounded when we move to higher-order structures like time evolving graphs, since finding the rank of three-dimensional tensor is NP-hard [7]. Generating realistic data where the rank of the data is known can further promote research in developing and testing methods AutoTen [8] or NSVD [9] for approximating the rank of a tensor. We reserve this for future works, but it serves as a compelling motivation for the generation of realistic matrices and higher-order tensors of known rank.

In this work, we begin this journey by focusing on generating graphs (matrices) of fixed rank in this work because finding their rank is computationally tractable. This allows us to evaluate the effectiveness of our bound on the rank of the graph. We propose a new model, BRGAN-A, that first generates *factor* matrices of rank of at most k and uses these factor matrices to construct the adjacency matrix. This allows us to bound the rank of the generated graphs. We also propose another variant, BRGAN-B that lets us generate graphs of *exactly* rank k. To do this, we propose a novel penalty term on the loss function of the generator that encourages the

generation of orthogonal factor matrices.

We evaluate these models and show that they has competitive performance to current state-of-the-art neural network graph generation models despite a bound on the rank of the graph. We also evaluate the effectiveness of this bound and the sensitivity of our model to changes to k.

Our contributions include:

- **Novel problem formulation**: we propose the problem of adversarially generating an adjacency matrix with a desired rank.
- **Novel architecture**: we propose two novel architectures for the above problem that generate the adjacency matrix with a rank bound of k from factor matrices.
- **Novel regularization term**: we propose a novel regularization term that *increases* the rank of generated adjacency matrices up to k.
- **Extensive experimentation**: we thoroughly evaluate the effectiveness of our approach along three dimensions: the rank of the generated graphs, the realism of the generated graphs, and the sensitivity of our model to hyperparameters.

II. Problem Formulation and Proposed Method

Table I provides a summary of the notation used in this work:

Notation	Meaning
$\mathbf{X}$, $\mathbf{x}$, x	Matrix, vector, scalar
$\|\mathbf{X}\|_F$	Frobenius norm
$\|\mathbf{x}\|_p$	ℓ_p-norm
$\mathbf{D}_\mathbf{M}$	Matrix of the diagonal entries of $\mathbf{M}$
k	The desired rank of the output matrices.
λ_k	Penalty term
$\mathrm{vec}(\mathbf{X})$	Vectorization operator (vertically stack the columns of $\mathbf{X}$ into a vector)

Table I: Table of symbols and their description.

We define the rank of a graph as the rank of its adjacency matrix. Recall that the rank of a matrix is equal to the number of rank-1 matrices required to sum up to it. In this work, we generate an adjacency matrix of at most rank k.

Current graph generation models produce graphs of unbounded rank. This is because existing methods tend to either model the graph as a sequence or generate the adjacency matrix directly, which makes it difficult to bound the rank of the resulting matrix.

The SVD can be used to find a rank-k approximation of an adjacency matrix which is optimal with respect to the 2-norm and Frobenius norm of the difference between the approximation and original matrix. However, this approximation will not necessarily preserve graph properties. Because of this, we propose BRGAN: a novel architecture that uses a bilinear network to generate an adjacency matrix from its factor matrices. This allows us to bound the rank of the output.

We propose 2 methods to do this, which we denote BRGAN-A and BRGAN-B.

A. BRGAN-A

BRGAN-A generates the adjacency matrix from two factor matrices, and allows for an upper bound on the rank of the generated matrices. One advantage of this over BRGAN-B is that it requires fewer parameters and has fewer restrictions on the generated factor matrices, which allows for more realistic graphs.

Let the two factor matrices be denoted $\mathbf{A}$ and $\mathbf{B}$, where $\mathbf{A} \in \mathbb{R}^{n\times k}$ and $\mathbf{B} \in \mathbb{R}^{k\times n}$. Then, their product $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{n\times n}$ and $\mathrm{rank}(\mathbf{C}) \leq \min(\mathrm{rank}(\mathbf{A}), \mathrm{rank}(\mathbf{B}))$. By definition, $\mathrm{rank}(\mathbf{A}) \leq \min(n, k)$ and $\mathrm{rank}(\mathbf{B}) \leq \min(n, k)$. If we define k such that $k \leq n$, we know that $\mathrm{rank}(\mathbf{A}), \mathrm{rank}(\mathbf{B}) \leq k$. Then, $\mathrm{rank}(\mathbf{C}) \leq \min(k, k) \leq k$. We exploit this property to generate a graph of bounded rank. Note that this approach does not guarantee that $\mathbf{C}$ is of exactly rank k because the vectors in $\mathbf{A}$ and $\mathbf{B}$ are not guaranteed to be linearly independent. However, we describe how to reduce the chance of this occurring in Section II-C.

In the case of an undirected graph, the adjacency matrix is symmetric. Our method can be easily adapted to naturally enforce this symmetry. We can generate a symmetric matrix $\mathbf{S}$ of rank k or lower from our factor matrix $\mathbf{A}$ if we take $\mathbf{S} = \mathbf{A}^T\mathbf{A}$. Then $\mathrm{rank}(\mathbf{S}) = \mathrm{rank}(\mathbf{A}) \leq k$. We can also incorporate our second factor matrix $\mathbf{B}$ if we let $\mathbf{S} = \mathbf{A}^T\mathbf{A} + \mathbf{B}^T\mathbf{B}$. This would lead to $\mathrm{rank}(\mathbf{S}) \leq 2k$. However, we leave a detailed evaluation of this approach to a future work.

B. BRGAN-B

BRGAN-B generates the adjacency matrix from three factor matrices. Two of these matrices are penalized towards orthogonality, and the third matrix is a diagonal matrix.

Let the first two factor matrices be denoted $\mathbf{U}$ and $\mathbf{V}^T$. Then, the third factor matrix is denoted $\mathbf{\Sigma}$. These correspond to their namesakes in the Singular Value Decomposition (SVD), where we can decompose a given matrix $\mathbf{M}$ into $\mathbf{M} = \mathbf{U\Sigma V}^T$.

As such, we can construct our adjacency matrix from the components by simply taking the product of the factor matrices. By the definition of the SVD, we know that the rank of the matrix is equal to the number of singular values in the diagonal matrix $\mathbf{\Sigma}$. This allows us to provide an exact bound on the rank of the generated graphs.

However, one requirement of the SVD is that the $\mathbf{U}$ and $\mathbf{V}^T$ matrices must be orthogonal. To do this, we introduce a novel penalty term on the orthogonality of the generated $\mathbf{U}$ and $\mathbf{V}^T$ matrices. We talk more about this term below in Section II-C.

It is also worth noting that this method could be easily extended to work for generating tensors of known rank like time-evolving graphs or multigraphs by applying the same idea to the CP (CANDECOMP/PARAFAC) decomposition [10]. However, this is beyond the scope of this paper, and we leave a detailed evaluation of this approach to a future work.

C. Rank Penalty

In the case of BRGAN-A, the generator provides a hard upper bound on the rank of the generated adjacency matrices,

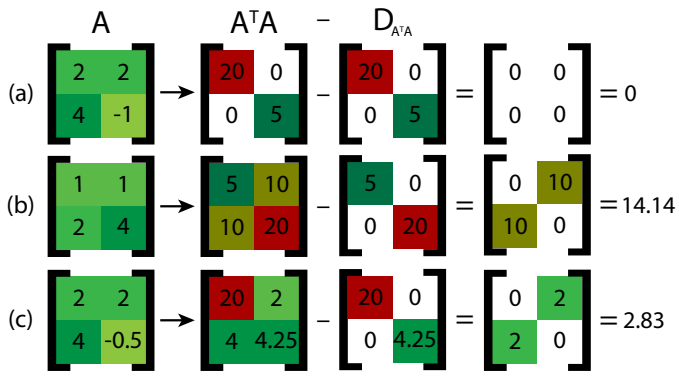

Figure 1: Example of penalty term 2 applied to 3 sample matrices. **(a)** shows that a matrix of orthogonal column vectors has a penalty of 0. **(b)** shows that a matrix of non-orthogonal column vectors has a non-zero penalty. **(c)** shows that a matrix that is almost column orthogonal has a relatively low penalty. Penalty term 1 can be viewed in a similar manner.

but does not offer a lower bound on the generated rank. This is because of the natural occurrence of linearly dependent columns in $\mathbf{A}$ and linearly dependent rows in $\mathbf{B}$. While this can be beneficial for some use cases, having as many matrices of exactly rank k as possible may also be desirable.

BRGAN-B allows for an exact bound on the rank of the generated adjacency matrices, but also requires that $\mathbf{U}$ and $\mathbf{V}^T$ be orthogonal. While an orthogonal matrix can be obtained from a given matrix through Gram–Schmidt orthogonalization, it requires that all the vectors in the matrix be linearly independent—i.e., the matrix is non-singular. Since it is possible for the factor matrices to be singular, we can cannot use this method.

To solve this issue, we propose two possible novel penalty terms on the loss function of the generator, whose goal is to promote orthogonality among the columns of $\mathbf{A}$ and the rows of $\mathbf{B}$.

$$\mathcal{L} = \mathcal{L}_{\mathcal{G}} + \lambda_k \left(\left\| \text{vec}\left(\mathbf{I} - \mathbf{A}^T\mathbf{A}\right) \right\|_p + \left\| \text{vec}\left(\mathbf{I} - \mathbf{B}\mathbf{B}^T\right) \right\|_p \right) \tag{1}$$

$$\mathcal{L} = \mathcal{L}_{\mathcal{G}} + \lambda_k (\left\| \text{vec}\left(\mathbf{A}^T\mathbf{A} - \mathbf{D}_{\mathbf{A}^T\mathbf{A}}\right) \right\|_p + \left\| \text{vec}\left(\mathbf{B}\mathbf{B}^T - \mathbf{D}_{\mathbf{B}\mathbf{B}^T}\right) \right\|_p) \tag{2}$$

Penalty term 1 encourages generating normalized orthogonal vectors and is used primarily for BRGAN-B, but also works well for BRGAN-A. Penalty term 2 encourages non-normalized orthogonal column vectors in $\mathbf{A}$ and row vectors in $\mathbf{B}$.

We know that if some matrix $\mathbf{M}$ is orthonormal, then $\mathbf{M}^T = \mathbf{M}^{-1}$ and $\mathbf{M}^T\mathbf{M} = \mathbf{M}\mathbf{M}^T = \mathbf{I}$. Term 1 penalizes that difference. While this is useful for BRGAN-B, where the matrices must be strictly orthonormal to obey the SVD formulation, sometimes we do not need such a strict constraint.

The intuition behind term 2 can be explained if we view the matrix multiplication as a sum of outer products of the matrices. In BRGAN-A, we define the output matrix $\mathbf{C} = \mathbf{A}\mathbf{B}$, and in BRGAN-B, we define $\mathbf{C} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. In the case of BRGAN-B, $\mathbf{\Sigma}$ simply scales the values and therefore does not affect the rank of the resulting matrix as long as it consists of non-zero entries.

Therefore, we can analyze the multiplication as $\mathbf{C} = \mathbf{A}\mathbf{B}$, where $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}$ and $\mathbf{B} = \mathbf{V}^T$ in the case of BRGAN-B. Then, we can rewrite this product in terms of the outer products of column vectors, where $\mathbf{a}_i$ denotes the i-th column of $\mathbf{a}$ and $\mathbf{b}_i^T$ denotes the i-th row of $\mathbf{b}$:

$$\mathbf{C} = \mathbf{A}\mathbf{B} = \sum_{i=1}^{n} \mathbf{a}_i\mathbf{b}_i^T \tag{3}$$

Since our goal is to make $\mathbf{C}$ full rank (or as high rank as possible), we would therefore ideally want the columns of $\mathbf{A}$ and the rows of $\mathbf{B}$ to be linearly independent (within themselves). Recall that this is possible since $k \leq n$.

Term 2 is different from term 1 in that its goal is to encourage only column-wise orthogonality in $\mathbf{A}$ and row-wise orthogonality in $\mathbf{B}$. This provides a lighter constraint than that of Term 1 since the rows of $\mathbf{A}$ and the columns of $\mathbf{B}$ do not have to be orthogonal. This also means that they do not have to be normalized. This is sufficient when using the BRGAN-A architecture, as the aim is simply to reduce the number of linearly dependent vectors in $\mathbf{C} = \mathbf{A}\mathbf{B}$, not to enforce strict matrix orthogonality.

All of the terms are written with the generic ℓ_p-norm, since we found that using the ℓ_1 and ℓ_2 norms can lead to slightly different results as shown in Table VI. Since we use the vectorization operator, note that $p = 2$ is equivalent to calculating the Frobenius norm of the matrix.

Using the ℓ_1-norm has the benefit of encouraging sparsity in the diagonal of the Gram matrices, which contain pairwise inner products of the rows and columns of $\mathbf{A}$ and $\mathbf{B}$, and should ideally be zero. In contrast, using the ℓ_2-norm encourages minimizing the squared distance in the values. However, we found that both give similar results, but the ℓ_2 norm is more stable. The reason for this is that using the ℓ_1-norm can result in sparse Gram matrices, which encourages the generator to generate close-to-zero factor matrices. This is especially prevalent with term 2.

λ_k controls the strength of our regularization, although we have found that it works well even with small values of λ_k. We evaluate the performance of this penalty term and its sensitivity to different values of λ_k in Section IV-D below.

III. Architecture

A GAN consists of two main models: the generator $\mathcal{G}$ and the discriminator $\mathcal{D}$. The generator maps a sample from a latent space z to a graph G. The discriminator takes an adjacency matrix $\mathbf{M}$ and outputs the probability p of a sample

being fake. These two models are then trained in unison in the hope that the generator will improve at generating realistic graphs and the discriminator will improve in identifying fake graphs.

One limitation of the original GAN model [3] is that it works poorly for discrete values because it uses the Jensen-Shannon (JS) divergence. WGAN [11] helps mitigate the effects of this by using Wasserstein distance instead. WGAN-GP [12] further improves the training of WGANs by applying a gradient penalty and CT-GAN [13] further improves on WGAN-GP with the addition of a consistency term. Fan and Bert [4] also found that the CT-GANs [13] perform well in graph generation. We found similar results in our testing and therefore use the CT-GAN framework for BRGAN.

A. Generator Architecture

1) BRGAN-A: The BRGAN-A generator consists of a multi-layer perceptron (MLP) network followed by two MLP networks. The first MLP network acts as a feature extractor on the noise vector z and the MLP networks use it to generate the factor matrices. This significantly reduces the number of parameters, and we found that it converges more quickly and performs better than without using the shared layer.

We apply $\tanh$ to the output of each network to clip values to $[0, 1]$. While this does not cause each entry in the final adjacency matrix M to be bounded to $[0, 1]$, it does help reduce the chance of the discriminator easily detecting fakes based on the value of the nodes, rather than on the structure of the graph. This helps ensure that the model continues training.

2) BRGAN-B: The BRGAN-B generator is similar to that of BRGAN-B, with the addition of another MLP network to generate the vector of singular values. The major difference between BRGAN-A is that it also requires the use of a penalty term to enforce the orthonormality of $\mathbf{U}$ and $\mathbf{V}^T$. Without this, the rank of the generated graphs will not be bounded correctly.

B. Discriminator Architecture

The BRGAN discriminator accepts an $n \times n$ adjacency matrix and consists of several Graph Convolutional Network (GCN) [5] layers followed by a fully-connected layer and a sigmoid. The benefit of using a GCN for the discriminator is that GCNs are permutation invariant. We use residual GCN connections by performing a max pool across all the GCN layer outputs. As Fan and Huang [4] found, the network performs better with these residual connections. BRGAN-A and BRGAN-B share the same discriminator architecture.

We compare the performance of the discriminator with two different architectures below in Table II.

IV. Experimental Evaluation

In evaluating our model, we try to answer three different questions. First, can our model generate graphs of a specific rank? Second, do those graphs mimic real-world graphs? Finally, is our rank penalty term effective?

	Small CORA			Small Citeseer		
Model Name	Deg	Clust	Orbit	Deg	Clust	Orbit
FC-BRGAN ($k = 15$)	0.28	0.51	0.09	0.27	0.20	0.11
FC-BRGAN ($k = 25$)	0.44	0.67	0.24	0.11	0.40	0.06
FC-BRGAN ($k = 35$)	0.33	0.82	0.18	0.13	0.42	0.14
BRGAN ($k = 15$)	0.08	**0.32**	**0.05**	0.10	0.19	**0.05**
BRGAN ($k = 25$)	**0.06**	0.34	0.08	0.11	**0.18**	0.08
BRGAN ($k = 35$)	0.11	0.33	0.09	**0.05**	0.26	0.06

Table II: Comparison of a GCN-based BRGAN discriminator and a fully-connected (FC-BRGAN) discriminator. We can see that using a GCN results in increased realism, even across different values of k. See Section IV-C for a detailed description of the evaluation criteria.

A. Methodology

We evaluated our model using egonets extracted from the CORA and Citeseer citation graph datasets [14]. Each dataset was split into a small dataset, which consists of 2-egonets and 3-egonets with $[30, 50]$ nodes, and a large dataset, which consists of 3-egonets with $[150, 200]$ nodes. We chose to use these datasets because they were also used to evaluate LGGAN [4], which allows us to benchmark our results against theirs. Note that while these are the same base datasets as in You *et al.* [2], we sample the egonets differently since we also wish to compare our results with Fan and Huang [4].

While BRGAN's discriminator uses a GCN and is node permutation invariant, BRGAN's generator is affected by node ordering because it uses a MLP. As such, we use the approach used by You *et al.*[2] and Fan *et al.*[4] and generate all possible BFS orderings of the graph. This allows use to only have n^2 permutations per graph rather than the full $n!$ possible permutations. We then train on this augmented dataset.

Both the generator and discriminator were trained with the RMSprop [15] optimizer because we found that it tends to be more stable than Adam [16] for this case. Arjovsky *et al.*[11] also suggested using RMSprop for WGANs, although Wei *et al.*[13] uses ADAM. We did not perform a hyperparameter search for the learning rate and found that our model generally works well with any reasonable learning rate. For the CTGAN loss function, we used $\lambda_1 = 10$ and $\lambda_2 = 2$—the same values as Wei *et al.*[13].

The output of $\mathcal{G}$ is an adjacency matrix M with values in $[0, r]$, although values tend to be in the range $[0, 1]$ due to the input values also being in this range. The graphs are then thresholded by some threshold τ. That is:

$$\mathbf{M}_{i,j} = \begin{cases} 0, & \text{if } \mathbf{M}_{i,j} \leq \tau \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

τ controls the sparsity of the final graph. As $\tau \to 0$, $|V| \to 0$ and as $\tau \to r$, $|V| \to m \cdot n$. For our tests, we chose $\tau = 0.5$ because the majority of the values are in the range $[0, 1]$ and $\tau = 0.5$ provides a mid-way bound.

B. Generating graphs of a specific rank

Singular Value Decomposition (SVD) can be used to find the approximate rank of a matrix. The SVD of a matrix $\mathbf{M}$

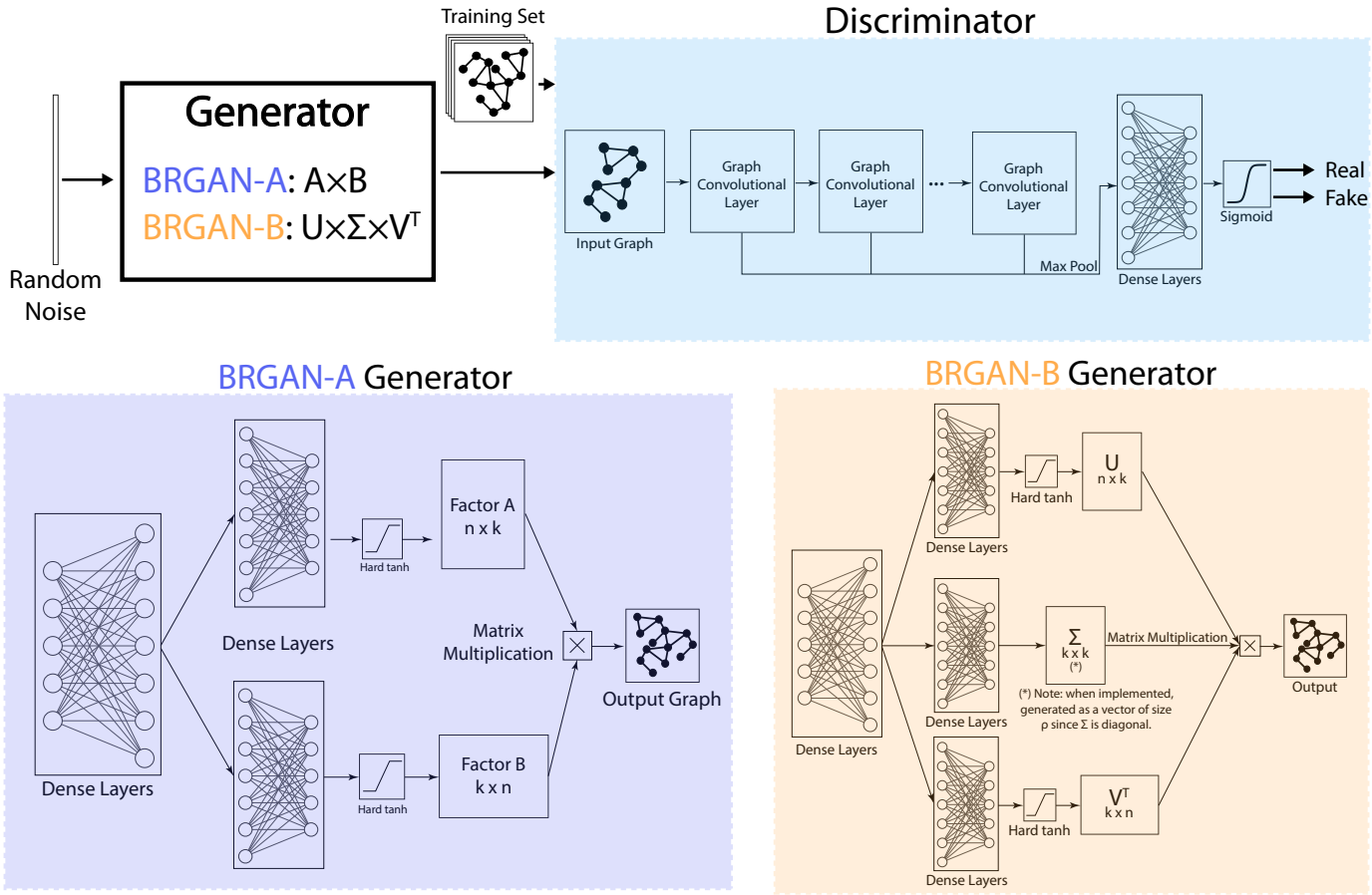

Figure 2: BRGAN architecture with both the BRGAN-A and BRGAN-B generators shown.

is $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U}$ is a $m \times m$ orthogonal matrix, $\mathbf{\Sigma}$ is a diagonal non-negative $m \times n$ matrix, and $\mathbf{V}$ is a $n \times n$ orthogonal matrix. By convention, the singular values $\sigma_1, \sigma_2, \ldots \sigma_m$ are sorted in descending order by value. Then, the approximate rank of a matrix can be found by finding the smallest index k where $\sigma_{k+1} < \epsilon$, where ϵ is some small constant. In our experiments, we found that $\epsilon = 10^{-6}$ is sufficient.

1) BRGAN-A: As theoretically expected, we found that our bound was effective and the rank of generated matrices were equal to or less than k. We also found that as k increases, the median rank of the generated graphs generally also increases. However, the median rank of the generated matrices is closer to k when k is small. This means that k becomes a tighter bound on the true rank of the generated graphs as $k \to 1$.

This is because it is difficult to create a realistic approximation of a high-rank matrix while maintaining low rank. However, as the rank bound is increased, the median generated rank does not increase linearly. This is because some of the input graphs are already of low rank and therefore some of the generated graphs will not be of higher rank, leading to a lower median generated graph rank.

We also noticed that the MMD of our graph attributes do not always uniformly improve as k increases. This is likely because that, as the number of parameters increases as k increases. However, the clustering MMD does improve in almost all cases as k increases, likely because of the reasons described in Section IV-C below.

2) BRGAN-B: BRGAN-B resulted in graphs of *exactly* rank k. This makes sense because the SVD formulation was used to generate the output graphs, which allows us to fully control the rank of the generated graphs as long as the rank penalty term is effective in orthonormalizing $\mathbf{U}$ and $\mathbf{V}^T$.

However, this does come at a cost in terms of the realism of the results as described in Section IV-C below.

C. Realism of the generated graphs

To evaluate the realism of the generated graphs, we compared several graph statistics of the input graphs with those of the output graphs. We did this by calculating the Mean Maximum Discrepancy (MMD) [17] between the degree distribution, clustering coefficient, and orbits of the two sets of graphs. We compared this to the results of various other graph generation methods in the tables below. The source code for LGGAN was not available at the time of writing, so we used the results reported in their paper [4].

1) BRGAN-A: While the results vary with the value of k, we can see that BRGAN-A generally has similar degree and orbit MMDs when compared to LGGAN. However, it tends to have a higher clustering MMD. This is likely because

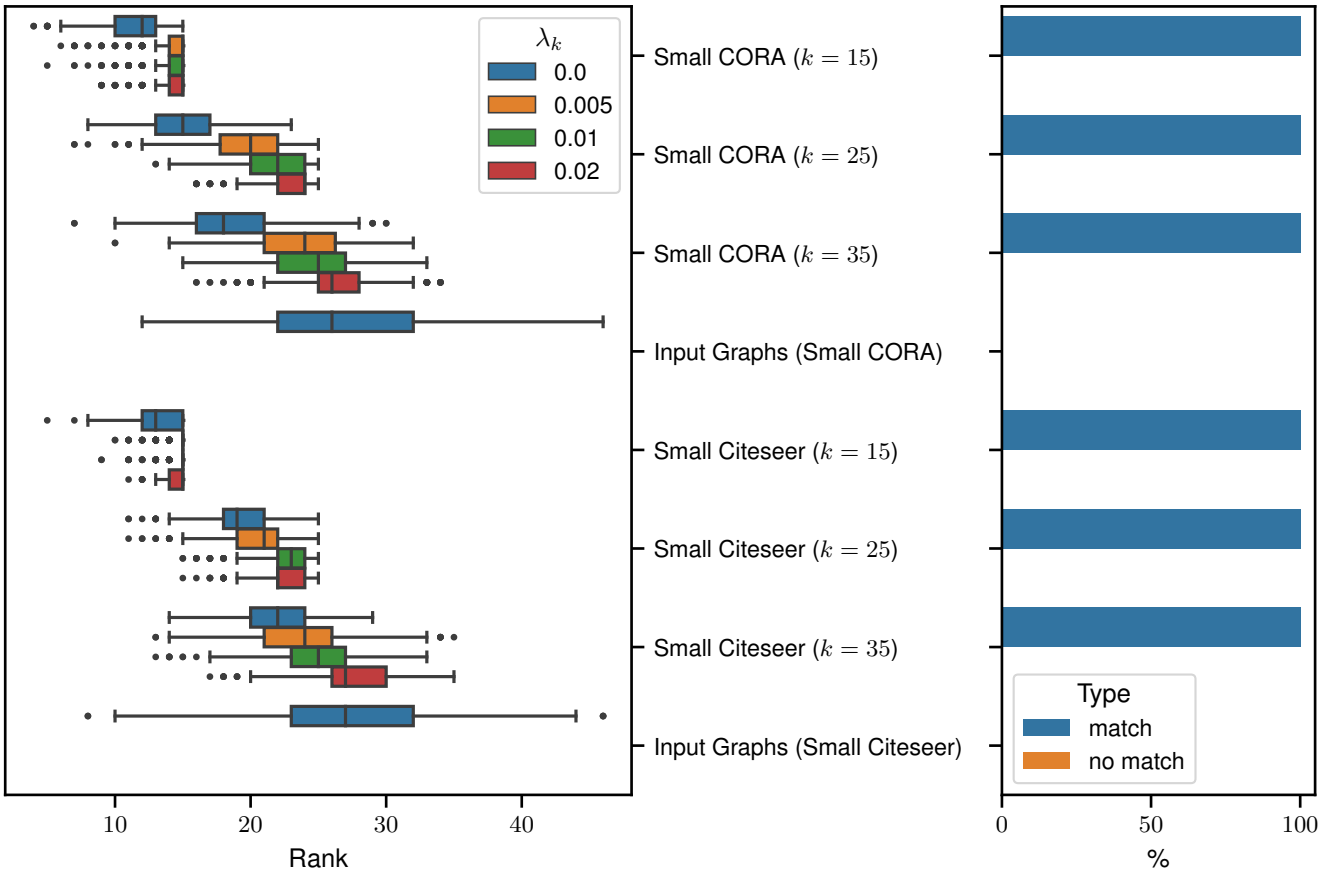

Figure 3: Left: distribution of the rank of generated graphs from the small CORA and Citeseer datasets for different values of k and λ_k with BRGAN-A. We can see that increasing λ_k is relatively effective in raising the rank of the generated graphs. **Right:** % of graphs matching/not matching rank k for BRGAN-B. In all of the experiments, all of the generated graphs were *exactly* of rank k.

lowering rank has the largest effect (out of the three tested graph statistics) on the clustering coefficient of a graph. This is because of the fact that we are imposing low-rank structure on the adjacency matrix of the graph. Then, all of the nodes live in a much lower dimension, which increases the density of connections and therefore would also increase the clustering coefficient.

We tested this by quantitatively calculating the MMD between graph statistics of a rank-k approximation of the graph (calcuating using SVD) and the original graph. The highest MMD value from this test was the one for the clustering coefficient. We observed this behavior in both the CORA and Citeseer datasets.

2) BRGAN-B: The MMD scores were significantly worse than that of BRGAN-A. This is expected, however, since we artificially inflate the rank of all the generated graphs to values higher than that of the input distribution. The graph attribute that is most affected is the degree distribution, which is drastically different from that of the input graph. The effect of this is less apparent as k gets closer to the mode of the ranks of the input graphs.

This is because new values have to be added to the matrix to artificially increase the rank of the generated graphs and each such new value constitutes a new edge. Each such case increases the degree of two nodes, further skewing the degree distribution.

This restriction also seems to result in BRGAN-B generating similar graphs across different values of k. This is why the MMD scores on all of the BRGAN-B models are similar.

D. Effectiveness of Rank Penalty Term

From the left side of Fig. 3 we can observe that the rank penalty term is effective on BRGAN-A, and as λ_k increases, the rank of the generated graphs also increases while still being bounded by k. On the right side, we can see that *all* of the graphs generated by BRGAN-B are of exactly rank k. Without the penalty term, the rank of the generated graphs are not bounded by k.

Table V show the performance of BRGAN-A with different values of λ_k and k. We can note see that generally as λ_k increases, model performance decreases. This is to be expected because a high value of λ_k causes the variance in rank of the generated graphs to decrease artificially, making it harder to model the graphs in the input dataset.

	Small CORA			Small Citeseer		
Model Name	Deg	Clust	Orbit	Deg	Clust	Orbit
Erdos-Renyi	0.68	0.94	0.48	0.63	0.86	0.12
Barabási–Albert	0.31	0.53	0.11	0.37	0.18	0.11
MMSB	0.21	0.68	0.07	0.37	0.18	0.11
DeepGMG	0.34	0.44	0.27	0.27	0.36	0.2
GraphRNN	0.26	0.38	0.39	0.19	0.2	0.39
LGGAN	0.13	**0.08**	**0.03**	0.17	**0.13**	**0.04**
BRGAN-A ($k = 10$)	**0.02**	0.64	0.07	0.08	0.18	0.12
BRGAN-A ($k = 15$)	**0.02**	0.56	0.04	0.11	0.22	0.08
BRGAN-A ($k = 20$)	0.03	0.47	0.07	0.07	0.23	0.08
BRGAN-A ($k = 30$)	0.09	0.39	0.11	0.09	0.28	0.07
BRGAN-A ($k = 50$)	0.07	0.34	0.11	**0.03**	0.27	0.05
BRGAN-B ($k = 10$)	1.32	0.17	0.75	1.40	0.30	0.85
BRGAN-B ($k = 15$)	1.35	0.17	0.78	0.81	1.35	0.30
BRGAN-B ($k = 20$)	1.38	0.18	0.80	1.40	0.28	0.86
BRGAN-B ($k = 30$)	1.41	0.18	0.84	1.41	0.28	0.88
BRGAN-B ($k = 50$)	1.41	0.18	0.85	1.43	0.30	0.92

Table III: Evaluation results for the small datasets for BRGAN-A and BRGAN-B. For BRGAN-A, the CORA degree and orbit MMD values are largely competitive with the other generative models, but LGGAN has a significantly lower clustering MMD value. BRGAN-B does poorly in terms of matching the degree distribution due to the strict rank restriction (see Section IV-C). Note that the numbers for other models are the ones reported by Fan and Huang [4].

	Large CORA			Large Citeseer		
Model Name	Deg	Clust	Orbit	Deg	Clust	Orbit
Erdos-Renyi	0.88	1.45	0.27	0.82	1.57	0.06
Barabási–Albert	0.54	1.06	0.16	0.32	1.04	0.08
MMSB	0.12	0.68	0.09	0.08	0.5	0.11
GraphRNN	0.2	0.46	0.11	0.2	1.15	0.14
LGGAN	**0.15**	**0.21**	0.06	0.25	**0.12**	0.06
BRGAN-A ($k = 140$)	0.30	0.46	0.29	0.70	1.80	0.06
BRGAN-A ($k = 170$)	0.22	0.45	**0.03**	**0.13**	0.80	0.06

Table IV: Evaluation results for the large Citeseer and large CORA dataset. Note that the numbers for other models are the ones reported by Fan and Huang [4]

	Small CORA			Small Citeseer		
Model Name	Deg	Clust	Orbit	Deg	Clust	Orbit
BRGAN-A ($k = 15, \lambda_k = 0$)	**0.02**	0.56	**0.04**	**0.11**	0.22	**0.08**
BRGAN-A ($k = 15, \lambda_k = 0.005$)	0.51	**0.22**	0.16	0.44	**0.12**	0.19
BRGAN-A ($k = 15, \lambda_k = 0.01$)	0.54	0.39	0.14	0.53	0.18	0.21
BRGAN-A ($k = 15, \lambda_k = 0.02$)	0.64	0.48	0.16	0.64	0.44	0.27
BRGAN-A ($k = 25, \lambda_k = 0$)	**0.06**	0.34	**0.08**	0.11	**0.18**	0.08
BRGAN-A ($k = 25, \lambda_k = 0.005$)	0.83	0.36	0.10	**0.05**	0.27	**0.06**
BRGAN-A ($k = 25, \lambda_k = 0.01$)	0.46	**0.33**	0.14	0.44	0.20	0.24
BRGAN-A ($k = 25, \lambda_k = 0.02$)	0.59	0.49	0.18	0.62	0.35	0.27
BRGAN-A ($k = 35, \lambda_k = 0$)	**0.11**	0.33	**0.09**	0.05	0.26	0.06
BRGAN-A ($k = 35, \lambda_k = 0.005$)	0.22	0.30	0.14	**0.02**	0.29	**0.03**
BRGAN-A ($k = 35, \lambda_k = 0.01$)	0.47	**0.27**	0.17	0.42	**0.19**	0.23
BRGAN-A ($k = 35, \lambda_k = 0.02$)	0.58	0.48	0.17	0.63	0.49	0.29

Table V: Comparison of performance of BRGAN-A on the small Citeseer and CORA datasets for different values of k and λ_k. We can see that performance generally decreases as λ_k increases, as expected.

V. Related Work

Graph generation has been widely studied, and many different models exist for this task. Traditional statisical models such as the Barabási–Albert [1] and exponential random graph models are created to model specific graph properties and tend to work well for only specific use cases.

One such example is the Erdös-Rényi-Gilbert random graph

	Small CORA			Small Citeseer		
Model Name	Deg	Clust	Orbit	Deg	Clust	Orbit
BRGAN-A (ℓ_1)	0.63	1.06	0.18	0.68	1.06	0.30
BRGAN-A (ℓ_2)	**0.06**	**0.44**	**0.09**	**0.08**	**0.22**	**0.06**

Table VI: Performance of BRGAN-A with ℓ_1 and ℓ_2 norms with penalty term 1. This was run with $k = 25$ and $\lambda = 0.001$.

model [18] [19]. The model is often referred to in the form $G(N, p)$ or $G(N, E)$, where N is the number of nodes, E is the number of edges, and p is the probability of an edge existing between two nodes. One of the key properties of this model is that expected number of neighbors for each node is the same [20]. However, this property is often not true of real-world graphs.

Several models attempt to address this limitation by attempting to vary node degree and mimic more realistic node distributions. Notably, the Barabási–Albert model [1] attempts to address this issue by using preferential attachment, which mimics the evolution of a scale-free graph over time and results in the creation of "hubs" in the graph.

The stochastic block model (SBM) [21] allows for the generation of graphs with a set number of communities. It takes in a partition of the vertices into communities and a matrix $\mathbf{P}$, with each entry $\mathbf{P}_{i,j}$ representing the probability of an edge existing between the ith and jth communities.

Another common family of statistical graph generation models are the exponential random graph models (ERGMs). An example of this are Markov graphs, for which Frank and Strauss [22] proved the probability distributions. This was later generalized by Wasserman and Pattison [23] in the form of p^* models. With Markov Chain Monte Carlo (MCMC) proposed by Hunter and Handcock [24], it is possible to estimate the parameters of ERGMs and generate similar graphs.

However, these statistical models all share the weakness that they attempt to model specific graph properties. Recently, advances in neural networks and deep learning have led the creation of several new models that learn directly from an input distribution of one or more graph(s). Unlike traditional models, these methods attempt to learn the structure of a graph rather than simply attempting to match specific attributes. Some notable methods include GraphRNN [2], NetGAN [25], MolGAN [26], and GraphVAE [27].

NetGAN [25] takes in a graph and learns the distribution of biased random walks using a LSTM, which allows it to easily scale to large graphs. MolGAN [26] uses a reinforcement learning objective with a reward network (in addition to the standard discriminator and generator) to generate realistic molecular graphs. GraphGAN [28] proposes a novel graph softmax function and learns the connectivity distribution over the vertices, but does not directly generate similar graphs. GraphVAE [27] uses a Variational Autoencoder (VAE) to generate graphs.

However, none of these models provide a method to bound the rank of the generated graph. BRGAN allows a user to eas-

ily bound the rank of the generated graphs while maintaining competitive performance.

VI. Conclusion

In this work, we propose two variants of the Bounded Rank GAN (BRGAN), which both generate graphs or rank equal to or lower than a hyperparameter k. However, BRGAN-A only provides an upper bound on the rank. We solve this by proposing a novel penalty term, that encourages the generation of graphs of exactly rank k and by proposing the BRGAN-B architecture. BRGAN-B results in graphs of almost exactly rank k. We discuss the merits and limitations of both approaches, and evaluate the effect of different hyperparameters. Finally, we thoroughly evaluate the performance of BRGAN and show that it has competitive performance with existing models and that it provides an effective bound on the rank of generated graphs.

VII. Acknowledgements

Research was supported by the National Science Foundation under CAREER grant no. IIS 2046086. This research was sponsored by the Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes not withstanding any copyright notation here on.

References

[1] R. Albert and A.-L. Barabasi, "Statistical mechanics of complex networks," 6 2001. [Online]. Available: http://arxiv.org/abs/cond-mat/0106096http://dx.doi.org/10.1103/RevModPhys.74.47

[2] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models," 2 2018. [Online]. Available: http://arxiv.org/abs/1802.08773

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," 6 2014. [Online]. Available: http://arxiv.org/abs/1406.2661

[4] S. Fan and B. Huang, "Labeled Graph Generative Adversarial Networks," 6 2019. [Online]. Available: http://arxiv.org/abs/1906.03220

[5] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," 9 2016. [Online]. Available: http://arxiv.org/abs/1609.02907

[6] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis, "All you need is Low (rank): Defending against adversarial attacks on graphs," in *WSDM 2020 - Proceedings of the 13th International Conference on Web Search and Data Mining*. New York, NY, USA: Association for Computing Machinery, Inc, 1 2020, pp. 169–177. [Online]. Available: https://dl.acm.org/doi/10.1145/3336191.3371789

[7] J. Håstad, "Tensor rank is NP-complete," *Journal of Algorithms*, vol. 11, no. 4, pp. 644–654, 12 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0196677490900146

[8] Papalexakis, Evangelos E, "Automatic unsupervised tensor mining with quality assessment," in *SIAM SDM*, 2016.

[9] Y. Tsitsikas and E. E. Papalexakis, " NSVD : N ormalized S ingular V alue D eviation Reveals Number of Latent Factors in Tensor Decomposition ," in *Proceedings of the 2020 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 1 2020, pp. 667–675.

[10] H. A. L. Kiers, "Towards a standardized notation and terminology in multiway analysis," in *Towards a standardized notation and terminology in multiway analysis*, 2000. [Online]. Available: https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/10.1002/1099-128X(200005/06)14:3%3C105::AID-CEM582%3E3.0.CO;2-I

[11] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 1 2017. [Online]. Available: http://arxiv.org/abs/1701.07875

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," 3 2017. [Online]. Available: http://arxiv.org/abs/1704.00028

[13] X. Wei, B. Gong, Z. Liu, W. Lu, and L. Wang, "Improving the Improved Training of Wasserstein GANs: A Consistency Term and Its Dual Effect," 3 2018. [Online]. Available: http://arxiv.org/abs/1803.01541

[14] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, 2008.

[15] G. E. Hinton, N. Srivastava, and K. Swersky, "Lecture 6a- overview of mini-batch gradient descent," *COURSERA: Neural Networks for Machine Learning*, 2012.

[16] D. P. Kingma and J. Lei Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION," Tech. Rep. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf

[17] A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample-problem," in *Advances in Neural Information Processing Systems*, 2007.

[18] P. Erdős and A. Rényi, "On random graphs," *Publicationes Mathematicae*, 1959.

[19] E. N. Gilbert, "Random Graphs," *The Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, 12 1959. [Online]. Available: http://projecteuclid.org/euclid.aoms/1177706098

[20] A. Goldenberg, A. X. Zheng, S. E. Fienberg, and E. M. Airoldi, "A survey of statistical network models," 12 2009. [Online]. Available: http://arxiv.org/abs/0912.5410

[21] K. Nowicki and T. A. Snijders, "Estimation and prediction for stochastic blockstructures," *Journal of the American Statistical Association*, vol. 96, no. 455, pp. 1077–1087, 9 2001. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1198/016214501753208735

[22] O. Frank and D. Strauss, "Markov Graphs," *Journal of the American Statistical Association*, vol. 81, no. 395, pp. 832–842, 9 1986. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01621459.1986.10478342

[23] S. Wasserman and P. Pattison, "Logit models and logistic regressions for social networks: I. An introduction to Markov graphs andp," *Psychometrika*, vol. 61, no. 3, pp. 401–425, 9 1996. [Online]. Available: http://link.springer.com/10.1007/BF02294547

[24] D. R. Hunter and M. S. Handcock, "Inference in Curved Exponential Family Models for Networks," *Journal of Computational and Graphical Statistics*, vol. 15, no. 3, pp. 565–583, 9 2006. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1198/106186006X133069

[25] A. Bojchevski and O. Shchur, "NetGAN: Generating Graphs via Random Walks," Tech. Rep., 2018. [Online]. Available: https://www.kdd.in.tum.de/netgan

[26] N. De Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," 5 2018. [Online]. Available: https://arxiv.org/abs/1805.11973

[27] M. Simonovsky and N. Komodakis, "GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11139 LNCS, pp. 412–422, 2 2018. [Online]. Available: http://arxiv.org/abs/1802.03480

[28] 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," 11 2017. [Online]. Available: http://arxiv.org/abs/1711.08267