Explore Deep Graph Generation

Xiuye Gu
Department of Computer Science
Stanford University
xiuyegu@stanford.edu

Abstract

Effective generative models on graphs can be useful in many fields. In this project, we explore deep graph generation from two directions: 1) via an adversarial process we use Convolutional Neural Networks (CNNs) to model the whole adjacency matrix directly after sorting the nodes; 2) we built upon the very recent Graph Recurrent Attention Networks (GRANs), proposed a graph completeness judger network and improved its attention mechanism. The first direction works on small graphs to some extent but fails to work on large graphs and we analyze its failure. For the second direction, experiments on the grid and protein datasets show that our improved version outperforms the original approach. Due to the inherent limits of auto-regressive models, though the first approach failed, for future work we think combining the advantages of the two directions is worthwhile and there is still a long way to go on deep graph generation.

1 Introduction

> *What I cannot create, I do not understand.*
>
> — Richard Feynman

Graphs/Networks are prevalent in science, nature, and technology. Now that we are able to extract networks from real life, how can we model and generate graphs/networks that are very likely to exist but we haven't discovered yet? The advantages of generating realistic graphs are at least twofold: 1) it is useful in downstream applications like drug design, study of proteins, *etc*.; 2) it provides better null models — the analysis of real-world networks often relies on null models, however, traditional random graph models cannot capture the complicated structures of real-world graphs.

From a broader perspective, deep generative models have been widely studied in computer vision and natural language processing. They are able to generate images/videos (Goodfellow et al., 2014; Wang et al., 2018) and texts/speeches (Oord et al., 2016). Despite applications in different fields, the most popular deep generative models fall into three categories: auto-regressive models, variational autoencoders (VAEs) (Kingma and Welling, 2013), and generative adversarial networks (GANs) (Goodfellow et al., 2014). For vision applications, GAN is the most successful among the three – recent studies (Brock et al., 2018; Karras et al., 2019) are able to generate high-resolution and high-quality photo-realistic images. For NLP, due to the sequential nature of language, its go-to building block, Recurrent neural network (RNN), is an auto-regressive model. There are also NLP research involve VAEs and GANs.

For graphs, the generation task is much harder than computer vision and NLPs: they do not have the grid structures like images, nor do they have sequential orders like texts/audios. The most similar task is deep generation of 3D point clouds, which also requires permutation invariance, but 3D point clouds do not have edges, *i.e*., they have much simpler topological structures.

In this project, we explore two of the above categories on deep graph generation: GANs and auto-regressive models. 1) Nodes in the graphs can be sorted according to their properties and the graph's

topological structure, *e.g.*, node degrees, k-cores, BFS and DFS orderings. If ideally, using some sorting strategy, each graph has a unique ordering of its nodes, then there is a bijection between graphs and their adjacency matrices. By generating adjacency matrices using convolutional neural networks (CNNs) and GANs, we are able to generate graphs like generating images. 2) A paper on the most recent conference (Liao et al., 2019) achieves state-of-the-art time efficiency and sample quality, which better captures the auto-regressive conditioning between the already generated and to-be-generated parts of the graph using Graph Neural Networks (GNNs) with attention. We found that their approach has a significant drawback: The network itself does not know when to stop generation and they simply sample the generated graph sizes according to the training distribution. So we propose a novel network completeness judger network to fill up this hole. Also, we noticed that their attention mechanism has several drawbacks and can be further improved.

We perform experiments on two datasets, the synthetic Grid and the real-world Protein (Dobson and Doig, 2003) dataset. Our second direction is very successful on both datasets qualitatively and quantitatively. Though the first approach works to some extent on small grid graphs, it does not work on large graphs. We ascribe this failure to the nature of CNNs: it's not that GANs cannot work on graphs, but because photo-realistic images and realistic adjacency matrices are very different by definition and our network uses the architecture of CNNs for image generation, so future work may need to design specific deep neural nets for GANs on graphs.

2 Related work

Our work is related to the Erdős-Rényi, (Erdős and Rényi, 1960), Small-World (Watts and Strogatz, 1998), and Kronecker random graph models (Leskovec et al., 2010) introduced in the lecture. Both aim to generate graphs that capture properties of real graphs. Deep generative models are usually more powerful in modeling complicated structures than these traditional hand-crafted random graph models.

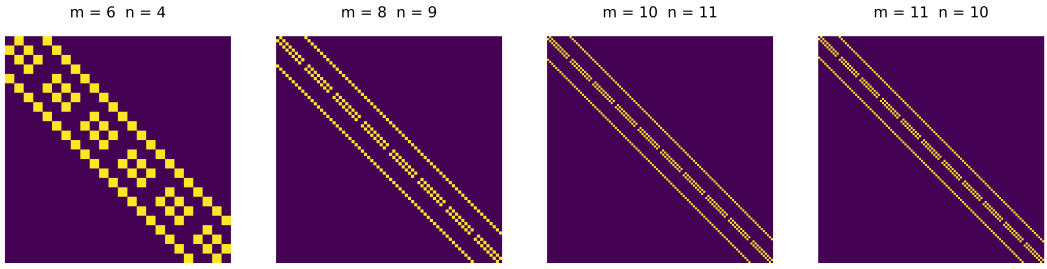
Simonovsky and Komodakis proposed a variational graph autoencoder: A stochastic graph encoder embeds real graphs into a latent space and given/sampling a point in this latent space, their graph decoder outputs a probabilistic fully-connected graph, from which discrete samples can be drawn. This approach falls into the VAE category. Our work pursues the same goal but adopts approaches from different categories.

GraphRNN (You et al., 2018) is a scalable framework lie in the autoregressive category. It is hierarchical: a graph-level RNN maintains the state of the graph and generates new nodes, and an edge-level RNN generates edges for each newly generated node. In order to improve scalability, they propose a BFS node ordering scheme. They also introduce novel evaluation metrics to measure graph generation quality quantitatively, which has been challenging. Our second approach is highly related to this work and lies in the same autoregressive category.

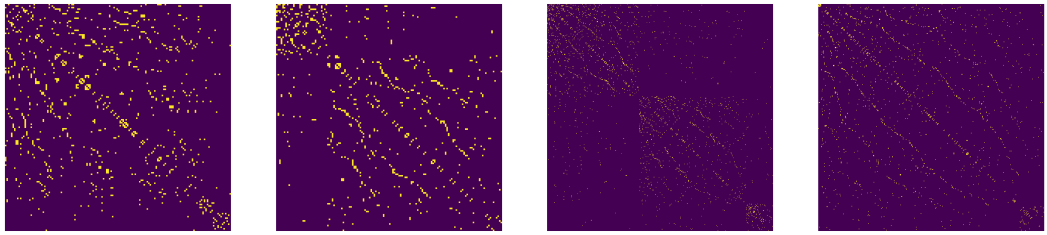
Our second approach builds upon (Liao et al., 2019), a very recent efficient and expressive autoregressive model that generates graphs one block of nodes and associated edges at a time. Compared to previous autoregressive models, their approach achieves better performance by adopting a GNN with attention for the generation step. The generation decisions of each block only depend on the current graph structure and thus they discard the graph-level RNN as well, in this way, their training is parallelized similar to PixelCNN (Van den Oord et al., 2016). They also use a parameterized mixture of Bernoulli instead of a single Bernoulli for output distributions to model correlations between generated edges. Our work further improves this method.

3 Datasets

Following (Liao et al., 2019), we conduct experiments on (1) Grid: we generate standard 2D grid graphs. For the second improved GRAN approach, we use $100 \leq |V| \leq 400$. For the first GAN approach, we experiment on smaller grids. It works with $4 \leq |V| \leq 30$ when the generated adjacency matrices have a resolution of 32×32 (recall that CNN GANs generate the adjacency matrices as 1-channel images and we use zero padding on graphs with $|V| < 32$), but it doesn't work on a slightly larger version: $4 \leq |V| \leq 64$ when the generated adjacency matrices have a resolution of 64×64. (2) Protein (Dobson and Doig, 2003): it contains 918 protein graphs with $100 \leq |V| \leq 500$, where nodes are amino acids and an edge exists between two nodes if they are less than 6 Angstroms away.



(a) Grid dataset using the default ordering. m, n stands for the size of each grid.



(b) Protein dataset using the descending k-core ordering and ties are broken by descending node degree ordering.

Figure 1: Plots of adjacency matrices after sorting the nodes. The adjacency matrix plots have obvious patterns on both datasets.

For both datasets, we randomly split the graphs into 80% and 20% subsets for training and testing. Example graphs of the two datasets are shown in Fig. 4 and 5.

4 First direction: CNN GANs on adjacency matrices

We argue that by ordering the nodes based on their properties, we can greatly reduce the number of possible orderings associated with each graph. And the resulting adjacency matrices should have some characteristics indicating that they belong to the associated graph distribution. If we can learn a generative model that outputs adjacency matrices with such characteristics, we may sample from this graph distribution.

To verify our argument, we plot the adjacency matrices after sorting the nodes by proper orderings, see Fig. 1. For the grid dataset, there is a bijection between graphs and their adjacency matrices after ordering. We observe an interesting and obvious pattern among these matrices: only the diagonal region has 1-entries, and the diagonal pattern has m sub-patterns where each sub-pattern consists of $2(n-1)$ 1-entries (m, n stands for the size of the grid). The protein dataset is more complicated, but it still has some salient characteristics: the upper left corner and the lower right corner is dense while the other two corners are sparse; the patterns near the diagonal are similar for all plots.

The adjacency matrices can be regarded as binary images, so we try to use CNN GANs for image generation to generate adjacency matrices. The motivation is that CNNs are much faster than autoregressive models and they have larger receptive field in higher layers to capture global information, unlike RNNs (You et al., 2018) that suffer from bottlenecks in handling long-term dependencies. And this approach supports directed graphs. However, adjacency matrices are very different from images, simply applying a CNN GAN for image generation fails easily and thus we adopt some special techniques.

Generative adversarial network. The general idea of generative adversarial network (Goodfellow et al., 2014) is to adopt a generator (G) and a discriminator (D), where the discriminator learns to differentiate the real samples and the generated samples, while the generator learns to sample from the distribution and to fool the discriminator, which forms a two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))].$$

As a common practice, the training has two stages, at the generator's stage, we fix the parameters of D and only optimize G; in the discriminator's stage, we fix the parameters of G and only optimize D.

We use LSGAN (Mao et al., 2017) instead of the vanilla GAN loss to stabilize training:

$$\min_D V_{LSGAN}(D) = \frac{1}{2}\mathbb{E}_{x\sim p_{data}(x)}[(1-D(x))^2] + \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[D(G(z))^2],$$
$$\min_G V_{LSGAN}(G) = \mathbb{E}_{z\sim p_z(z)}[(1-D(G(z)))^2].$$

Soft labeling and binary thresholding. Real adjacency matrices consist of all binary entries, while CNN generators use a Tanh layer to constrain the output within $(-1, 1)$, so CNN generators cannot output exact binary values. The gradient also saturates when the output approaches -1/1. Hence, we propose to adopt soft labeling on the real adjacency matrices before sending them into the discriminator. We denote the soft labeling function as $S(x)$ and $\epsilon \sim Uniform(0, 1)$ (a different ϵ is sampled for every x):

$$S(x) = \begin{cases} (1-\alpha)+\alpha\epsilon, & \text{if } x == 1 \\ -((1-\alpha)+\alpha\epsilon), & \text{otherwise (x == 0)} \end{cases}$$

where α is a hyper parameter. The main idea is to replace the exact 1s and 0s with some real numbers, but we still enforce a gap between the soft 1s and 0s.

An important difference between GANs and autoregressive models is that GANs take a random noise vector as input to model the random sampling process and their outputs are deterministic, while existing autoregressive models (You et al., 2018; Liao et al., 2019) do not have random inputs and they treat the outputs as probabilities. The latter is suboptimal: we do not have full access to the graph distribution, and thus we cannot model the probabilities correctly — we give zero/small probabilities to the data points that we never see during the training process, but these data points can come from the graph distribution that we want to model.

We find that binary thresholding works well on converting the real-valued outputs into binary adjacency matrices: if an entry is larger than the pre-defined threshold, then we convert it to 1 and to 0 otherwise.

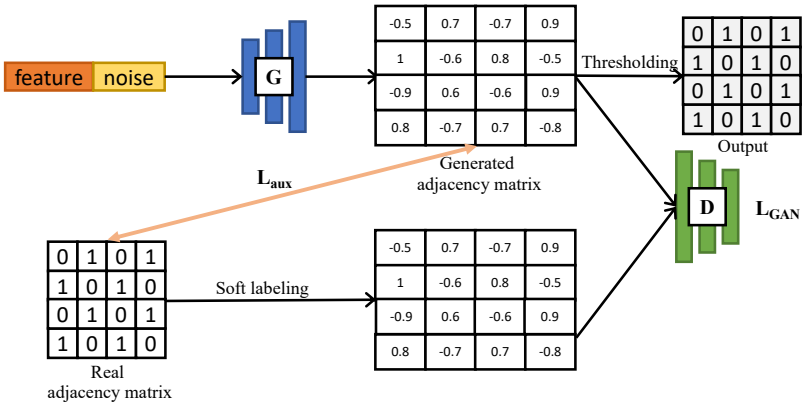


Figure 2: CNN GAN Network Architecture.

Network architecture. The overall network architecture is shown in Fig. 2. Our generator is based on the generator of DCGAN (Gao et al., 2018) and our discriminator is based on PatchGAN (Isola et al., 2017).

Vanilla GANs suffer from mode collapse (the generator tends to output the same simple but realistic graph), which achieves low loss but is not ideal. Inspired by InfoGAN (Chen et al., 2016), we add a feature vector f in addition to the random vector z as the input to G, *e.g.*, in grid graphs, we use the grid size (m, n) as the feature vector (we encode the integer vector as a one-hot vector). We then add an auxiliary loss $\mathcal{L}_{aux}$ to enforce the generated matrix to capture such features, *i.e.*, to increase mutual information between the input and the generation.

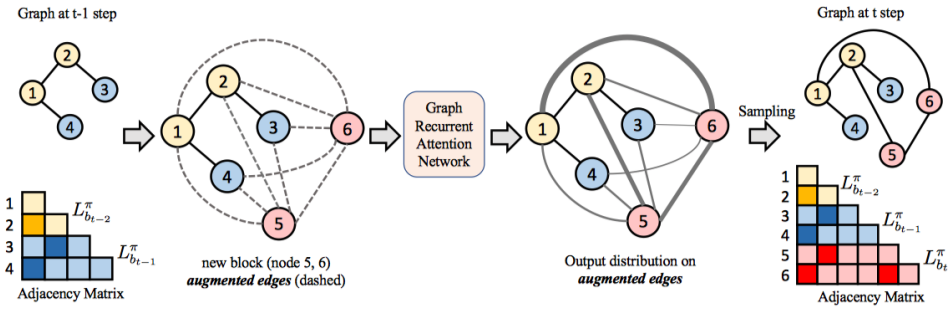


Figure 3: Overview of GRAN. Image taken from (Liao et al., 2019).

We keep a buffer that stores previously created graphs. Our inputs to the discriminator are random samples from this buffer so that the discriminator will not forget previous generations too soon and will be more robust. We show more details in the appendix.

5 Second approach: Improving GRAN

Fig. 3 shows the high-level idea of Graph Recurrent Attention Networks (GRANs) (Liao et al., 2019). It generates a block of nodes and the associated edges at a time. The new nodes are initialized to connect all existing nodes (*augmented edges*) and they use edge features to indicate whether the edges are augmented. The current augmented graph is fed into a Graph Neural Network (GNN) to predict the actual probabilities of the existence of the new edges. More specifically, the output of the last GNN layer (raw outputs) is then used to predict the final edge probabilities, which is modeled as a mixture of Bernoulli distributions. The importance of each Bernoulli distribution is the output of a multi-layer perceptron (MLP) taking the raw outputs as inputs. The probability for each Bernoulli is the output of another MLP also taking the raw outputs as inputs.

5.1 Graph completeness judger network

One advantage of GRAN is that because the generation decisions for each block only depend on the current graph structure, it enjoys parallel training. As a consequence, GRAN actually only learns to generate associated edges given an existing subgraph and a block of new nodes to be added. They use a hack to generate graphs of different sizes: they form a multinomial distribution of the graph size from the statistics of the training set. During inference time, they first sample a graph consists of N_{max} nodes, where N_{max} is the pre-defined maximum number of nodes. They then sample a number N from the aforementioned multinomial distribution and only keep the induced subgraph of the first N nodes.

This hack is limited for the sampling of graph size is totally unaware of the generated graph structure. As shown in the generated graphs (Fig. 4), though many graphs in the training set have N nodes, given the sampled graph of size N_{max}, the induced subgraph on its first N nodes may not necessarily lie in the graph distribution.

Hence we propose a graph completeness judger network to decide whether the induced subgraph lies in the graph distribution. One design choice is to train it jointly with the generation process so that the generation can stop at proper sizes, however, an induced subgraph can be either a good sample or a subgraph of another good sample. Take the Grid dataset as an example, graph with grid size $(m-1, n-1)$ can be a grid graph or the generation can continue as grid $(m, n), ...$ are also good samples. In this case, it's very hard to model the probability of whether to continue. Instead, we train a separate GNN which predicts how likely a graph lies in the distribution, similar to the idea of discriminators in GANs. We then form a categorical distribution of graph sizes and the probability for each size N is proportional to the output score of the completeness judger on the first N nodes' induced subgraph. In particular, our completeness judger network have a 3-layer GCN, which is followed by a global max pooling layer, and finally a 2-layer MLP.

There are far more unlikely graphs then likely graphs. We assume that GRAN is good enough to sample graphs of the desired distribution (it just doesn't know when to stop), so our negative samples

only come from subgraphs of the real graphs. To avoid imbalanced dataset, for each real graph, we randomly sample a single impossible subgraph on the fly that has graph size similar to the real graph.

5.2 Improve attention mechanism

The r-th round of message passing in the GNN for the generation step of GRAN is implemented as follows:

$$m_{ij}^r = f(h_i^r - h_j^r), \tag{1}$$

$$\tilde{h}_i^r = [h_i^r, x_i], \tag{2}$$

$$a_{ij}^r = \text{Sigmoid}\big(g(\tilde{h}_i^r - \tilde{h}_j^r)\big), \tag{3}$$

$$h_i^{r+1} = \text{GRU}(h_i^r, \sum_{j \in \mathcal{N}(i)} a_{ij}^r \otimes m_{ij}^r), \tag{4}$$

where h_i^r is the hidden state for node i at round r, and m_{ij}^r is the message vector from node i to j. x_i is a binary mask indicating whether node i is in the existing nodes (in which case all entries of x_i are 0s) or in the new block of B nodes (x_i is a one-hot vector of size B). a_{ij}^r is the attention weight associated with message m_{ij}^r. Both the message function f and the attention function g are implemented as 2-layer MLPs. The aggregation function is summation and the updating function is a GRU.

The $a_{ij}^r m_{ij}^r$ is implemented as element-wise multiplication, which can be rewrite as

$$\text{Sigmoid}\big(g(\tilde{h}_i^r - \tilde{h}_j^r)\big) \otimes f(h_i^r - h_j^r)).$$

It is very similar to the attention mask in (Pumarola et al., 2018), where the attention mask finds the locations on the image that need to be edited to change face expression. Though this scheme works well on image editing, we doubt that it works well on our task: 1) It breaks the learned correlation between the different dimensions of the same feature by weighting them differently. 2) It fails to utilize graph structure. Both the attention function and the message function only takes a single edge's feature as input. 3) It may be redundant. When we ignore the sigmoid function and when the g and f are linear layers, it can be simplified as $g(\tilde{h}_i^r - \tilde{h}_j^r)$.

All the three points can be addressed by design ideas similar to Graph Attention Networks (GATs) (Veličković et al., 2017). To address 1), we propose to use a shared weight for all dimensions and to use multi-head attention to improve modeling capacity. Dealing with 2), we can use the masked attention mechanism and use the softmax function to gather information within each local neighborhood, and it solves 3) as well. We also replace the subtractions ($h_i^r - h_j^r$ and $\tilde{h}_i^r - \tilde{h}_j^r$) with concatenations. It's less powerful since features from the source and target nodes share the same weights. Our improved GRAN is:

$$m_{ij}^{r,k} = f_k([h_i^r, h_j^r]), \tag{5}$$

$$\tilde{h}_i^r = [h_i^r, x_i], \tag{6}$$

$$a_{ij}^{r,k} = \text{Masked\_Softmax}\big(g_k([\tilde{h}_i^r, \tilde{h}_j^r])\big) = \frac{\exp\big(g_k([\tilde{h}_i^r, \tilde{h}_j^r])\big)}{\sum_{l \in \mathcal{N}_i} \exp\big(g_k([\tilde{h}_i^r, \tilde{h}_l^r])\big)}, \tag{7}$$

$$h_i^{r+1} = \text{GRU}\Big(h_i^r, \|_{k=1}^K \big(\sum_{j \in \mathcal{N}(i)} a_{ij}^{r,k} m_{ij}^{r,k}\big)\Big). \tag{8}$$

$\|$ denotes concatenation and we have K heads. Different from the original version, $a_{ij}^{r,k}$ is a scalar rather than a vector of the same dimensionality as m_{ij}^r.

6 Experiments

Evaluation metrics. Following (You et al., 2018), we compare the distributions of graph statistics between generated and real-world graphs. We use degree distribution, clustering coefficient distribution, and the number of occurrence of all orbits with 4 nodes as graph statistics. The difference

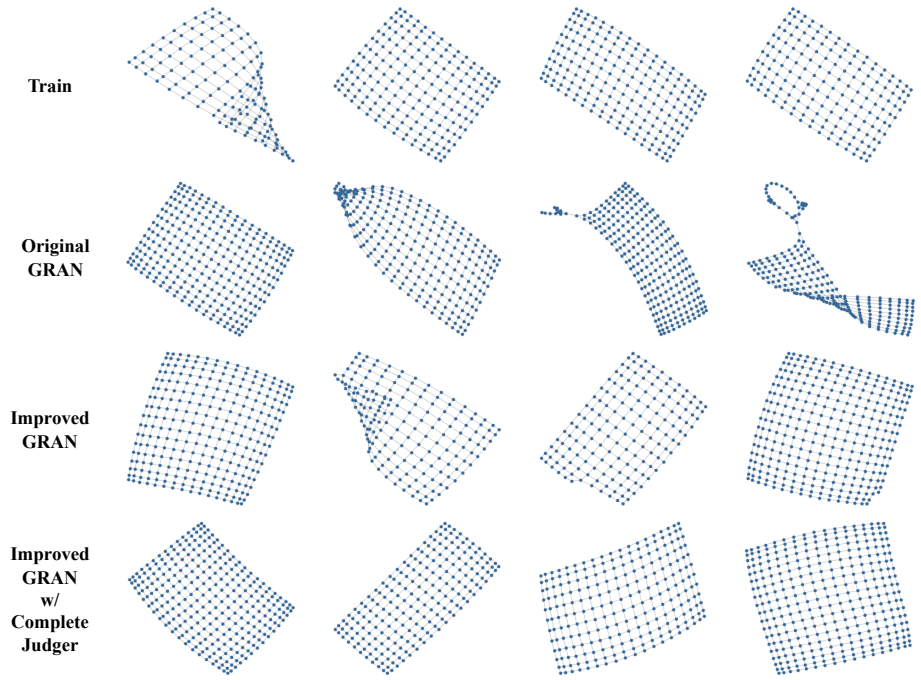


Figure 4: Qualitative results on the Grid dataset. Our improved version has better sample quality but the last three generations are incomplete grids, and our proposed complete judger fixes this problem.

	Grid				Protein			
	$\|V\|_{\max}$ = 361, $\|E\|_{\max}$ = 684 $\|V\|_{\text{avg}} \approx 210$, $\|E\|_{\text{avg}} \approx 392$				$\|V\|_{\max}$ = 500, $\|E\|_{\max}$ = 1575 $\|V\|_{\text{avg}} \approx 258$, $\|E\|_{\text{avg}} \approx 646$			
	Deg.	Clus.	Orbit	Spec.	Deg.	Clus.	Orbit	Spec.
GRAN	$6.84e^{-4}$	**0**	$1.45e^{-3}$	$1.50e^{-2}$	$6.91e^{-3}$	$9.30e^{-2}$	$7.75e^{-2}$	$4.74e^{-3}$
Improved	$1.42e^{-4}$	**0**	$2.41e^{-4}$	$\mathbf{8.48e^{-3}}$	$\mathbf{6.57e^{-3}}$	$\mathbf{7.44e^{-2}}$	$\mathbf{5.92e^{-2}}$	$\mathbf{4.71e^{-3}}$
w/ Judger	$\mathbf{4.33e^{-6}}$	$2.71e^{-5}$	$\mathbf{3.19e^{-6}}$	$1.35e^{-2}$	-	-	-	-

Table 1: Quantitative results on the Grid and Protein dataset. For all metrics, the smaller the better. -: not applicable due to lack of domain knowledge. Deg.: degree distribution, Clus.: clustering coefficients, Orbit: the number of 4-node orbits, Spec.: spectrum of graph Laplacian. The results demonstrate that our proposed improvements and the completeness judger network are effective.

between two distributions is evaluated by the maximum mean discrepancy (MMD). Following (Liao et al., 2019), since computing the MMD with the Gaussian EMD kernel is slow, we use the total variation (TV) distance and it is consistent with EMD. We also perform a spectral comparison by computing the eigenvalues of the normalized graph Laplacians and quantizing them to make an approximate discrete distribution. Unlike the other three metrics, the spectral comparison is able to capture global graph information. In our experiments, we found that these evaluation metrics may not work very well when the errors are subtle. Since evaluation on graph generation has been challenging, future work may propose better evaluation metrics, *e.g.*, training a discriminator-like network to judge whether the generated graph is sampled from the given distribution, but using this approach, how can we train and validate that the network is a good judger with good generalization ability becomes a new problem. We use the same hyper-parameters as (Liao et al., 2019). Unlike Liao et al., we train all models for 3,000 epochs.

6.1 Effectiveness of GRAN improvements

We evaluate our improved GRAN on both the Grid and Protein datasets. Due to lack of domain knowledge, we do not train a completeness judger network on the Protein dataset. As long as we have

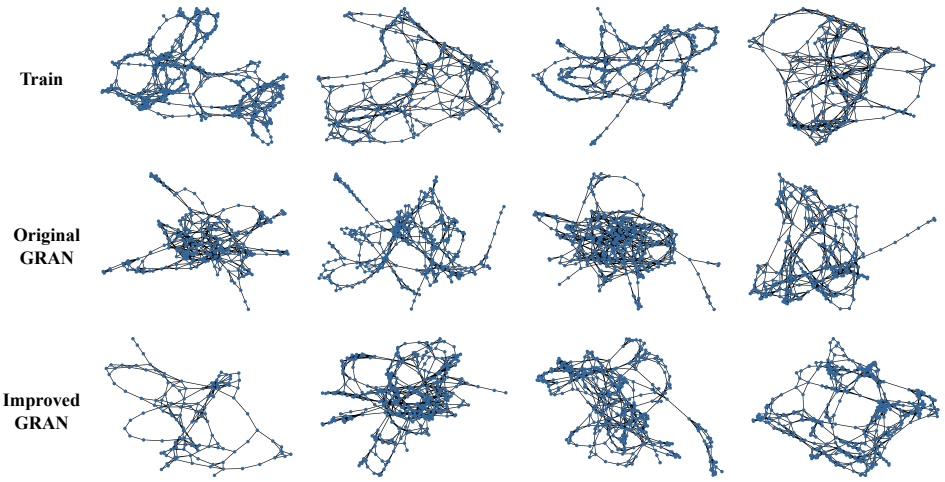


Figure 5: Qualitative results on the Protein dataset. Though it's hard to judge sample quality by visual inspection on this complicated dataset, we observe the generated graphs capture certain properties of the real graphs.

some rules or enough annotations, we are able to apply the judger on the Protein dataset in exactly the same manner.

We show qualitative results of the Grid dataset in Fig. 4, where our improved version outperforms the original version greatly. The only remaining problem is that the generated grids are incomplete. Our completeness judger works as expected and with its help, the created graphs become complete. The MMD measures in Table 1 also verifies the effectiveness of our proposed approaches.

Interestingly, we observe that using hard thresholding rather than random sampling according to the output scores produces visually better results, especially for the original GRAN. And it makes sense for the Grid dataset, for given a subgraph and a new node under the default ordering, the edges to be generated are mostly deterministic. But the quantitative results are worse (we report the random sampling results), which indicates that the evaluation scheme can be further improved, and the good quantitative results of random sampling should be attributed to the maximum likelihood training, though it does not necessarily relate to good sample quality. Anyway, the results have small absolute values in both cases, which may also indicate that when the metrics are low, they are no longer good in identifying better approaches.

We show some plots of the generation results of the Protein dataset in Fig. 5. From the plots we observe that our network learns to capture certain characteristics of the real-world protein graphs. The quantitative results in Table 1 demonstrate that our improved version outperforms the original version on all metrics.

6.2 Failure of CNN GANs on adjacency matrices

On the smaller grids and the generations have 32×32 resolution, the CNN GAN approach is able to generate perfect grids, but suffers greatly from mode collapse when we feed in only random noise as input, see Fig. 6. After adding the feature vector as input and imposing the $\mathcal{L}_{aux}$ to increase mutual information, the mode collapse is alleviated and the approach works to some extent (we include their qualitative results in the appendix).

However, this approach stops producing reasonable results when we switch to the slightly larger dataset and increase the generation resolution to 64×64. We observe that the generator's loss (the loss that makes the generator to fool the discriminator) does not decrease, while discriminator's loss (the loss that makes the discriminator differentiate real and fake) approaches 0 in a short time. This situation indicates that the discriminator is able to distinguish real adjacency matrices from generated ones very easily, and the generator cannot learn to generate when the discriminator is too powerful. The same situation happens even if we enforce an ℓ_1 loss between the output and the input adjacency

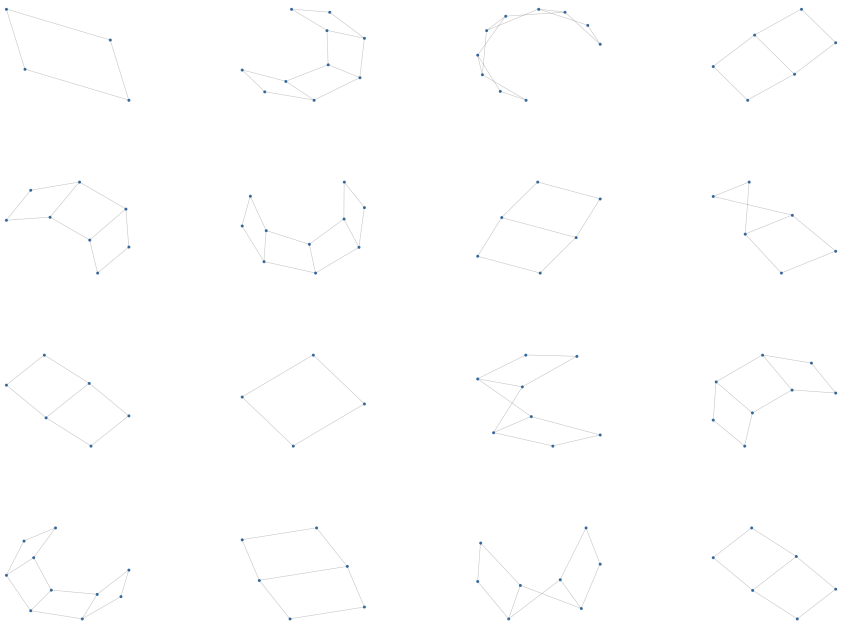


Figure 6: On the smallest Grid dataset, without feature input and the auxiliary loss $\mathcal{L}_{aux}$, the model only outputs perfect simple graphs but no complicated graphs, *i.e.*, it suffers from mode collapse, a common problem in GANs.

matrices (with a tolerance of errors less than α considering the soft labeling. The adjacency matrix for a grid graph is uniquely determined given its grid size under the default node ordering). We observe the ℓ_1 loss decreases to 0 while the G's loss remains large and the D's loss achieves 0 — even a single error in the output enables the discriminator to identity the generated matrices from real ones.

The most possible reason for the failure is: For generators that generate images from input vectors, they rely on either deconvolution or upsampling layers to gradually increase the generation's resolution. Both layers involve repetition: for upsampling, each pixel is replicated within its local neighborhood at the finer resolution; for deconvolution, the convolution weights are replicated at different pixel locations when the inputs are similar, *e.g.*, background pixels have the same color, and there is the well-known checkerboard artifacts. However, adjacency matrices are mostly sparse and the 1-entries usually appear without the aforementioned repetition patterns, which makes the CNN GANs fail when the resolution is higher — more upsampling/deconvolution layers. And for image generation, an error on a single pixel usually doesn't affect the generation quality, while an error on an entry of the adjacency matrix is very likely to cause the total failure of the graph generation.

We also considered using the reparameterization trick to sample from the generator's real-valued outputs while keeping the whole process differentiable. The only reparameterization trick I find that works for our distribution is the Gumbel-Softmax (Jang et al., 2016). But I observe that its behavior is strange across various temperature settings, so I do not adopt it. The TA's comment on our milestone suggests using a GNN as the discriminator, which requires differentiable binary values in the output adjacency matrices and thus requires reparameterization, so we do not explore this direction.

7 Discussion

In this project, we explore one GAN approach and one auto-regressive approach for deep graph generation. Though the GAN approach failed while the auto-regressive approach succeeded, we would like to point out that the auto-regressive approach models likelihood, which is not necessarily a good indicator of the sample quality. One possible next step is to use a GNN as the generator

network for GANs instead of CNNs. For example, we assume the graph is initially fully connected as in GraphVAE, and the GNN takes the number of nodes and a random noise vector as input. It then generates samples by gradually refining the modeling of the edges. In addition, InfoGAN (Chen et al., 2016) is a very useful approach in dealing with mode collapse by increasing mutual information and this idea can be easily adapted to GNNs.

Acknowledgments

We thank all the course staff for the hard work throughout the quarter and the generous Google cloud platform funding that let us explore state-of-the-art models.

Appendix

We show the network details of our generator and discriminator for the GAN approach in Table 2 and 3.

On the smallest Grid dataset, after adding the feature vector input, we found that it achieves better multi-modality when there is no noise input, see Fig. 7. Experiments show that it still has mode collapse problem when there is noise input of the same length as the feature input.

After adding the auxiliary loss, we obtain the results in Fig. 8. We achieve multi-modality with both the feature input and random noise input.

References

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Fei Gao, Yue Yang, Jun Wang, Jinping Sun, Erfu Yang, and Huiyu Zhou. A deep convolutional generative adversarial networks (dcgans)-based semi-supervised method for object recognition in synthetic aperture radar (sar) images. *Remote Sensing*, 10(6):846, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Layer	#channels	kernel	stride	padding
TransConv1	64	4	1	0
TransConv2	32	4	2	1
TransConv3	16	4	2	1
TransConv4	1	4	2	2

Table 2: Generator network details. Each TransConv layer except the first and last one is followed by an instance normalization layer and a ReLU Layer. After the last TransConv layer there is a final activation layer to make the output values lie in a desired range, for which we use a Tanh layer.

Layer	#channels	kernel	stride	padding
Conv1	16	4	2	0
Conv2	32	4	2	0
Conv3	64	4	1	0
Conv4	1	3	1	0

Table 3: Discriminator network details. Each Conv layer except the first and the last one is followed by an instance normalization layer and a LeakyReLU with a negative slope of 0.2.

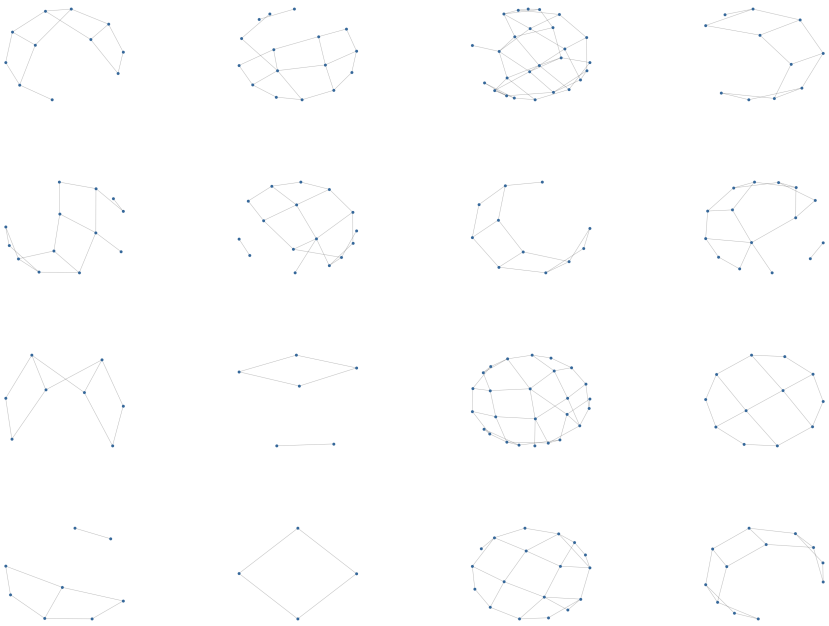


Figure 7: After incorporating the feature input, we found it achieves multi-modality when there is no noise input.

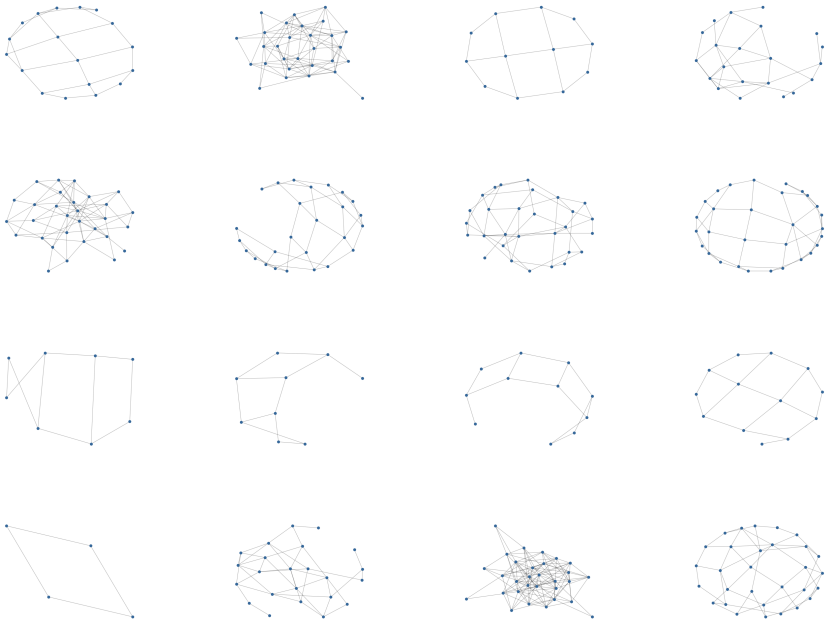


Figure 8: After imposing $\mathcal{L}_{aux}$ and incorporating the feature input, we found that it achieves multi-modality when there is noise input.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and*

pattern recognition, pages 1125–1134, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11 (Feb):985–1042, 2010.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pages 4257–4267, 2019.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Albert Pumarola, Antonio Agudo, Aleix M Martinez, Alberto Sanfeliu, and Francesc Moreno-Noguer. Ganimation: Anatomically-aware facial animation from a single image. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 818–833, 2018.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pages 412–422. Springer, 2018.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *nature*, 393 (6684):440, 1998.

Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.