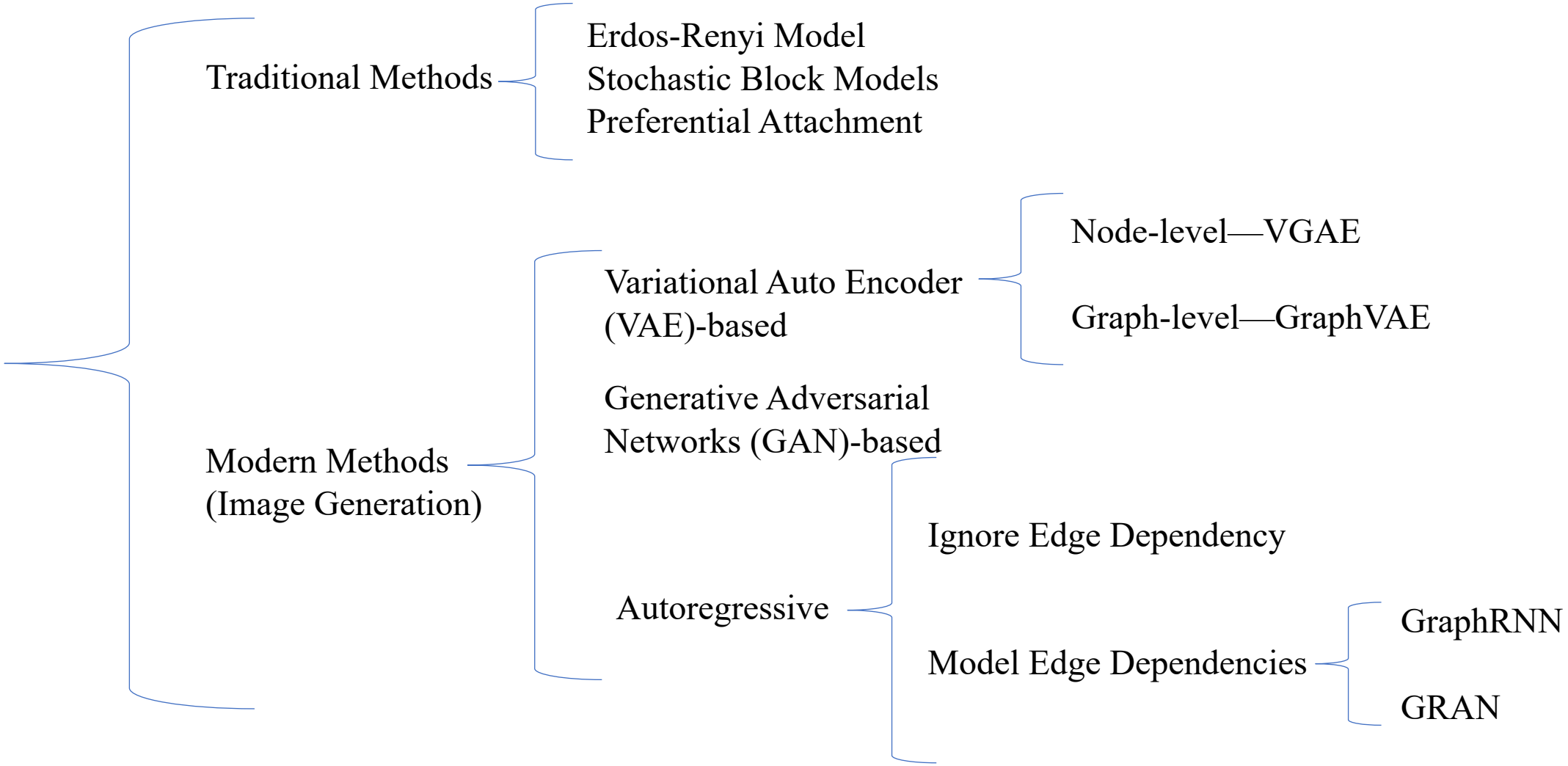
Generative Graph Models

Sitao Luan

Graph Generative Models



Traditional Graph Generation Approaches

We can frame this generative process as a way of specifying the probability or likelihood $P(\mathbf{A}[u, v] = 1)$ of an edge existing between two nodes u and v.

The generative process should be both tractable and able to generate graphs with non-trivial properties or characteristics.

Tractability is essential because we want to be able to sample or analyze the graphs that are generated (This is different with prediction tasks). However, we also want these graphs to have some properties that make them good models for the kinds of graphs we see in the real world.

Erdos-Renyi (ER) Model

- We define the likelihood of an edge occurring between any pair of nodes as

$$P(\mathbf{A}[u, v] = 1) = r, \forall u, v \in \mathcal{V}, u \neq v, \tag{8.1}$$

  where $r \in [0, 1]$ is parameter controlling the density of the graph. the time complexity to generate a graph is $O(|\mathcal{V}|^2)$.

- ER model is simple; however, it does not generate very realistic graphs. In particular, the only property that we can control in the ER model is the density of the graph. Other graph properties—such as the degree distribution, existence of community structures, node clustering coefficients, and the occurrence of structural motifs—are not captured by the ER model. It is well known that graphs generated by the ER model fail to reflect the distribution of these more complex graph properties, which are known to be important in the structure and function of real-world graphs.

Stochastic Block Models (SBMs)

- SBMs seek to generate graphs with community structure.

In a basic SBM model, we specify a number γ of different blocks: $\mathcal{C}_1, ..., \mathcal{C}_\gamma$. Every node $u \in \mathcal{V}$ then has a probability p_i of belonging to block i, i.e. $p_i = P(u \in \mathcal{C}_i), \forall u \in \mathcal{V}, i = 1, ..., \gamma$ where $\sum_{i=1}^{\gamma} p_i = 1$. Edge probabilities are then specified by a block-to-block probability matrix $\mathbf{C} \in [0, 1]^{\gamma \times \gamma}$, where $\mathbf{C}[i, j]$ gives the probability of an edge occuring between a node in block $\mathcal{C}_i$ and a node in block $\mathcal{C}_j$. The generative process for the basic SBM model is as follows:

1. For every node $u \in \mathcal{V}$, we assign u to a block $\mathcal{C}_i$ by sampling from the categorical distribution defined by $(p_i, ..., p_\gamma)$.

2. For every pair of nodes $u \in C_i$ and $v \in C_j$ we sample an edge according to

$$P(\mathbf{A}[u, v] = 1) = \mathbf{C}[i, j]. \tag{8.2}$$

Preferential Attachment (PA)

- SBM approach is limited in that it fails to capture the structural characteristics of individual nodes that are present in most real-world graphs, e.g., all nodes within a community have the same degree distribution. This means that the structure of individual communities is relatively homogeneous in that all the nodes have similar structural properties (e.g., similar degrees and clustering coefficients). Unfortunately, however, this homogeneity is quite unrealistic in the real world.

- In real-world graphs we often see much more heterogeneous and varied degree distributions, for example, with many low-degree nodes and a small number of high-degree “hub” nodes.

Preferential Attachment (PA)

- The PA model is built around the assumption that many real-world graphs exhibit power law degree distributions, meaning that the probability of a node u having degree d_u is roughly given by the following equation:

$$P(d_u = k) \propto k^{-\alpha}, \tag{8.3}$$

where $\alpha > 1$ a parameter.

Preferential Attachment (PA)

1. First, we initialize a fully connected graph with m_0 nodes.

2. Next, we iteratively add $n - m_0$ nodes to this graph. For each new node u that we add at iteration t, we connect it to $m < m_0$ existing nodes in the graph, and we choose its m neighbors by sampling without replacement according to the following probability distribution:

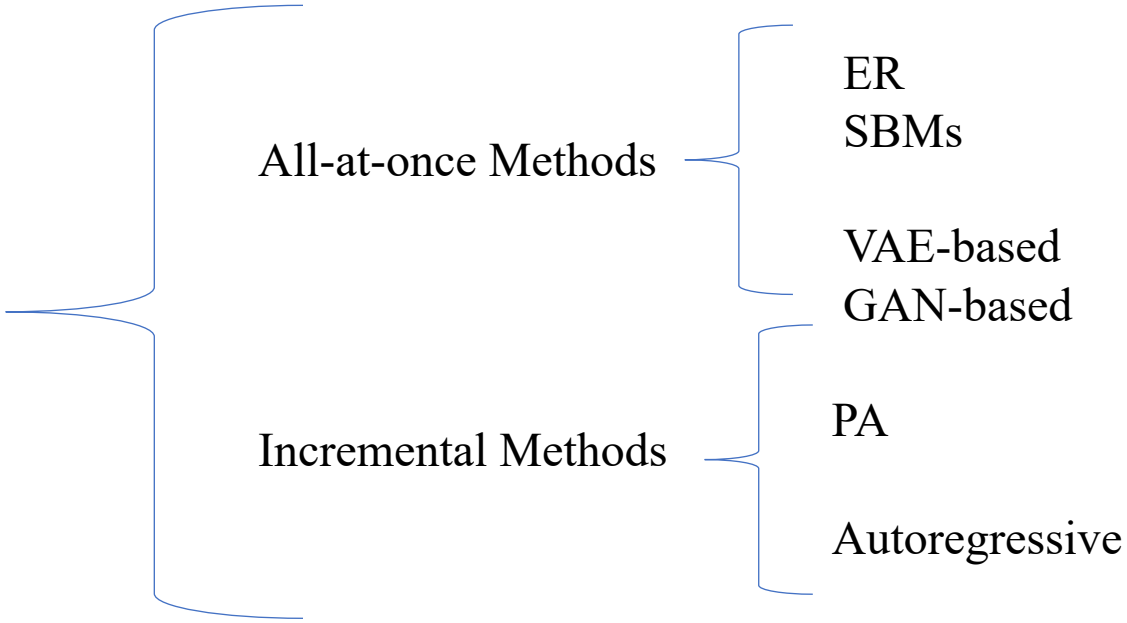
$$P(\mathbf{A}[u, v]) = \frac{d_v^{(t)}}{\sum_{v' \in \mathcal{V}^{(t)}} d_{v'}^{(t)}}, \tag{8.4}$$

where $d_v^{(t)}$ denotes the degree of node v at iteration t and $\mathcal{V}^{(t)}$ denotes the set of nodes that have been added to the graph up to iteration t.

PA model described above generates connected graphs that have power law degree distributions with $\alpha = 3$.

Modern Methods- Deep Generative Models

- The traditional approaches can generate graphs with several predefined parameters to guarantee some hand-coding properties, but they lack the ability to learn a generative model from a set of training graph data.



Sunday, March 21, 2021 11:09 PM

Suppose x is generated based on an unknown latent variable z. (generative process)

$$p(x) = \int_z p(x,z) = \int p(x|z)\,p(z)$$ time-consuming

now we want to infer the posterior of z given x, $p(z|x)$, but computational intractable

but we can approx $p(z|x)$ by $q(z|x)$, where $q(z|x)$ has a tractable form.

$$\min KL(q(z|x)\,||\,p(z|x))$$

$$= -E_q\left[\log \frac{p(z|x)}{q(z|x)}\right] = -E_q[\log p(z|x)] + E_q[\log q(z|x)]$$

$$= -E_q\left[\log \frac{p(z)\,p(x|z)}{p(x)}\right] + E_q[\log q(z|x)]$$

$$= -E_q\left[\log \frac{p(z)}{q(z|x)}\right] - E_q[p(x|z)] + E_q[\log p(x)]$$

$$= KL(q(z|x)\,||\,p(z)) - E_q(p(x|z)) + \log p(x)$$

$$= -ELBO(p(x)) + \log p(x)$$

VAE-based Methods

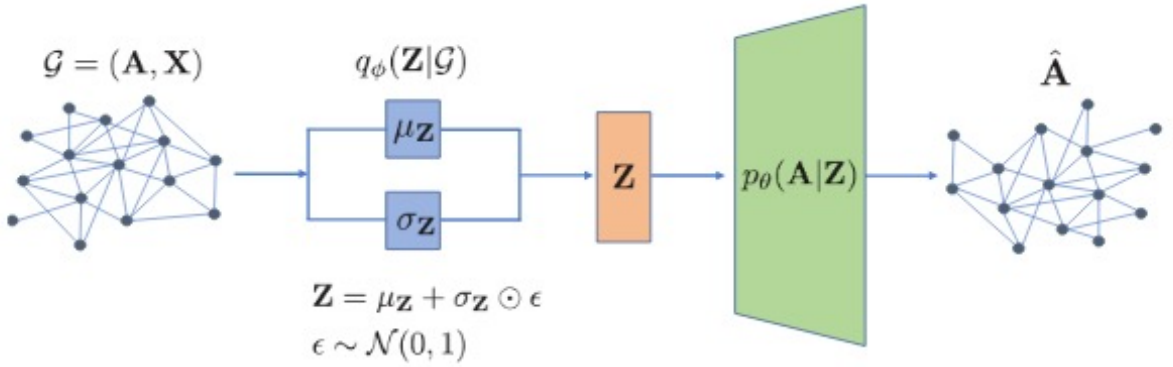


Figure 9.1: Illustration of a standard VAE model applied to the graph setting. An *encoder* neural network maps the input graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ to a *posterior distribution* $q_\phi(\mathbf{Z}|\mathcal{G})$ over latent variables $\mathbf{Z}$. Given a sample from this posterior, the *decoder* model $p_\theta(\mathbf{A}|\mathbf{Z})$ attempts to reconstruct the adjacency matrix.

Goal: train a probabilistic decoder model $p_\theta(\mathbf{A}|\mathbf{Z})$ from which we can sample realistic graphs (i.e., adjacency matrices) $\hat{\mathbf{A}} \sim p_\theta(\mathbf{A}|\mathbf{Z})$ by conditioning on a latent variable Z. In a probabilistic sense, we aim to learn a conditional distribution over adjacency matrices (with the distribution being conditioned on some latent variable).

We jointly train the encoder and decoder so that the decoder is able to reconstruct training graphs given a latent variable $Z \sim q_\theta(Z|G)$ sampled from the encoder. Then, after training, we can discard the encoder and generate new graphs by sampling latent variables $Z \sim p(Z)$ from some (unconditional) prior distribution and feeding these sampled latent variables to the decoder.

VAE-based Methods

1. A *probabilistic encoder* model q_ϕ. In the case of graphs, the probabilistic encoder model takes a graph $\mathcal{G}$ as input. From this input, q_ϕ then defines a distribution $q_\phi(\mathbf{Z}|\mathcal{G})$ over *latent representations*. Generally, in VAEs the *reparameterization trick* with Gaussian random variables is used to design a probabilistic q_ϕ function. That is, we specify the latent conditional distribution as $\mathbf{Z} \sim \mathcal{N}(\mu_\phi(\mathcal{G}), \sigma(\phi(\mathcal{G}))$, where μ_ϕ and σ_ϕ are neural networks that generate the mean and variance parameters for a normal distribution, from which we sample latent embeddings $\mathbf{Z}$.

2. A *probabilistic decoder* model p_θ. The decoder takes a latent representation $\mathbf{Z}$ as input and uses this input to specify a conditional distribution over graphs. In this chapter, we will assume that p_θ defines a conditional distribution over the entries of the adjacency matrix, i.e., we can compute $p_\theta(\mathbf{A}[u, v] = 1|\mathbf{Z})$.

3. A *prior distribution* $p(\mathbf{Z})$ over the latent space. In this chapter we will adopt the standard Gaussian prior $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$, which is commonly used for VAEs.

Given these components and a set of training graphs $\{\mathcal{G}_1, .., \mathcal{G}_n\}$, we can train a VAE model by minimizing the evidence likelihood lower bound (ELBO):

$$\mathcal{L} = \sum_{\mathcal{G}_i \in \{G_1, \ldots, \mathcal{G}_n\}} \mathbb{E}_{q_\theta(\mathbf{Z}|\mathcal{G}_i)}[p_\theta(\mathcal{G}_i|\mathbf{Z})] - \mathrm{KL}(q_\theta(\mathbf{Z}|\mathcal{G}_i)\|p(\mathbf{Z})). \tag{9.1}$$

VAE-based Methods

- The key intuition is that we want to generate a distribution over latent representations so that the following two (conflicting) goals are satisfied:

1. The sampled latent representations encode enough information to allow our decoder to reconstruct the input.

2. The latent distribution is as close as possible to the prior.

The first goal ensures that we learn to decode meaningful graphs from the encoded latent representations, when we have training graphs as input. The second goal acts as a regularizer and ensures that we can decode meaningful graphs even when we sample latent representations from the prior $p(Z)$.

we will describe two different ways in which the VAE idea can be instantiated for graphs.

Encoder of Variational Graph Autoencoder (VGAE)

- Given an adjacency matrix A and node features X as input, we use two separate GNNs to generate mean and variance parameters, respectively, conditioned on this input:

$$\mu_{\mathbf{Z}} = \mathrm{GNN}_{\mu}(\mathbf{A}, \mathbf{X}) \qquad \log \sigma_{\mathbf{Z}} = \mathrm{GNN}_{\sigma}(\mathbf{A}, \mathbf{X}). \tag{9.2}$$

Here, μ_Z is a $|V| \times d$ -dimensional matrix, which specifies a mean embedding value for each node in the input graph. The $\log\sigma_Z \in R^{|V| \times d}$ matrix similarly specifies the log-variance for the latent embedding of each node. Given the encoded μ_Z and $\log\sigma_Z$ parameters, we can sample a set of latent node embeddings by computing

$$\mathbf{Z} = \epsilon \circ \exp\left(\log(\sigma_{\mathbf{Z}})\right) + \mu_{\mathbf{Z}}, \tag{9.3}$$

where $\epsilon \sim N(0, \mathbf{1})$ is a $|V| \times d$ dimensional matrix with independently sampled unit normal entries.

Decoder VGAE

- Given a matrix of sampled node embeddings $Z \in R^{|V| \times d}$, the goal of the decoder model is to predict the likelihood of all the edges in the graph. VGAE employs a simple dot-product decoder defined as follows:

$$p_\theta(\mathbf{A}[u,v] = 1 | \mathbf{z}_u, \mathbf{z}_v) = \sigma(\mathbf{z}_u^\top \mathbf{z}_v), \tag{9.4}$$

Where σ is used to denote the sigmoid function. We simply assume independence between edges and define the posterior $p_\theta(G|Z)$ over the full graph as follows:

$$p_\theta(\mathcal{G}|\mathbf{Z}) = \prod_{(u,v) \in \mathcal{V}^2} p_\theta(\mathbf{A}[u,v] = 1 | \mathbf{z}_u, \mathbf{z}_v), \tag{9.5}$$

To generate discrete graphs after training, we can sample edges based on the posterior Bernoulli distributions in Equation (9.4).

Limitations of VGAE

- The generative capacity of this basic approach is extremely limited, especially when a simple dot-product decoder is used.
- Some variants of VGAE are proposed to make the decoder more powerful. Nonetheless, the simple node-level VAE approach has not emerged as a successful and useful approach for graph generation. It has achieved strong results on reconstruction tasks and as an autoencoder framework, but as a generative model, this simple approach is severely limited.

GraphVAE—Graph-level Latents

- we modify the encoder and decoder functions to work with graph-level latent representations (vector) $\boldsymbol{z}_G$.
- let GNN: $Z^{|V|\times|V|}\times R^{|V|\times m}\rightarrow R^{|V|\times d}$denote any k-layer GNN, which outputs a matrix of node embeddings, and we will use POOL: $R^{|V|\times d}\rightarrow R^{d}$ to denote a pooling function that maps a matrix of node embeddings $Z\in R^{|V|\times d}$to a graph-level embedding vector $\boldsymbol{z}_G\in R^{d}$. Then, the graph-level encoder is defined as

$$\mu_{\mathbf{z}_G}=\mathrm{POOL}_{\mu}\left(\mathrm{GNN}_{\mu}(\mathbf{A},\mathbf{X})\right)\qquad \log\sigma_{\mathbf{z}_G}=\mathrm{POOL}_{\sigma}\left(\mathrm{GNN}_{\sigma}(\mathbf{A},\mathbf{X})\right),\quad (9.6)$$

A single graph-level embedding is sampled from $\boldsymbol{z}_G\sim N(\mu_{\boldsymbol{z}_G},\sigma_{\boldsymbol{z}_G})$.

Decoder of GraphVAE

- we use an MLP to map the latent vector z_G to a matrix $\tilde{A} \in [0,1]^{|V|\times|V|}$ of edge probabilities:

$$\tilde{\mathbf{A}} = \sigma\left(\mathrm{MLP}(\mathbf{z}_{\mathcal{G}})\right), \tag{9.7}$$

where the sigmoid functionσis used to guarantee entries in $[0,1]$. we can then define the posterior distribution in an analogous way as the node-level case:

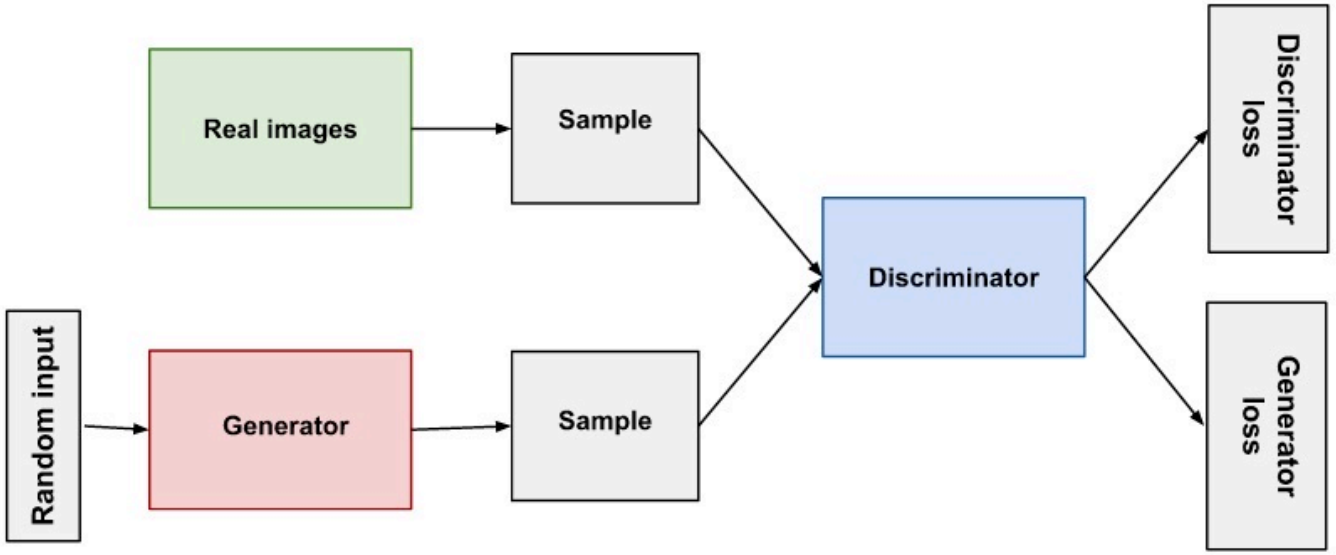
$$p_\theta(\mathcal{G}|\mathbf{z}_{\mathcal{G}}) = \prod_{(u,v)\in\mathcal{V}} \tilde{\mathbf{A}}[u,v]\mathbf{A}[u,v] + (1-\tilde{\mathbf{A}}[u,v])(1-\mathbf{A}[u,v]), \tag{9.8}$$

Where A denotes the true adjacency matrix of graph G and $\tilde{A}$ is our predicted matrix of edge probabilities.

Limitations of Graph-level VAE Approach

- If we are using an MLP as a decoder, then we need to assume a fixed number of nodes.
- The issue of specifying node orderings.

GAN-based Methods



- Basic idea behind GAN: First, we define a trainable generator network $g_\theta: R^d \rightarrow X$. This generator network is trained to generate realistic (but fake) data samples $x \in X$ by taking a random seed $z \in R^d$ as input (e.g., a sample from a normal distribution).
- At the same time, we define a discriminator network $d_\varphi: X \rightarrow [0,1]$. The goal of the discriminator is to distinguish between real data samples $x \in X$ and samples generated by the generator $x \in X$. Here, we will assume that discriminator outputs the probability that a given input is fake.

GAN-based Methods

- Both the generator and discriminator are optimized jointly in an adversarial game:

$$\min_{\theta} \max_{\phi} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log(1 - d_{\phi}(\mathbf{x}))] + \mathbb{E}_{\mathbf{z} \sim p_{\text{seed}}(\mathbf{z})}[\log(d_{\phi}(g_{\theta}(\mathbf{z}))], \qquad (9.10)$$

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \qquad (1)$$

Where $p_{data}(x)$ denotes the empirical distribution of real data samples (e.g., a uniform sample over a training set) and $p_{seed}(z)$ denotes the random seed distribution (e.g., a standard multivariate normal distribution).

The generator is attempting to minimize the discriminatory power of the discriminator, while the discriminator is attempting to maximize its ability to detect fake samples.

GAN-based Graph Generation

- For the generator, we can employ a simple multi-layer perceptron (MLP) to generate a matrix of edge probabilities given a seed vector z:

$$\tilde{\mathbf{A}} = \sigma\left(\mathbf{MLP}(\mathbf{z})\right), \tag{9.11}$$

- Given this matrix of edge probabilities, we can then generate a discrete adjacency matrix $\hat{A} \in Z^{|V|\times|V|}$ by sampling independent Bernoulli variables for each edge, with probabilities given by the entries of $\tilde{A}$; i.e., $\hat{A}\,[u, v] \sim Bernoulli(\tilde{A}[u, v])$. For the discriminator, we can employ any GNN-based graph classification model.

Benefits and Limitations

Benefits: Unlike the reconstruction loss used in VAE-based methods, GAN-based framework removes the complication of specifying a node ordering in the loss computation.

Limitations: GAN-based approaches to graph generation have so far received less attention and success than their variational counterparts. This is likely due to the difficulties involved in the minimax optimization that GAN-based approaches require, and investigating the limits of GAN-based graph generation is currently an open problem

Autoregressive Methods: Motivation

- Modeling Edge Dependencies: previous methods assume that edges were generated independently. From a probabilistic perspective, we defined the likelihood of a graph given a latent representation $\boldsymbol{z}$ by decomposing the overall likelihood into a set of independent edge likelihoods as follows:

$$P(\mathcal{G}|\mathbf{z}) = \prod_{(u,v)\in\mathcal{V}^2} P(\mathbf{A}[u,v]|\mathbf{z}). \tag{9.12}$$

Assuming independence between edges is convenient, as it simplifies the likelihood model and allows for efficient computations. However, it is a strong and limiting assumption, since real-world graphs exhibit many complex dependencies between edges

Autoregressive Methods

- In the autoregressive approach, we assume that edges are generated sequentially, and that the likelihood of each edge can be conditioned on the edges that have been previously generated. We use L to denote the lower-triangular portion of the symmetric adjacency matrix A. We will then use the notation $L[v_1, :]$ to denote the row of L corresponding to node v_1, and we will assume that the rows of L are indexed by nodes $v_1, \ldots, v_{|V|}$. Given this notation, the autoregressive approach amounts to the following decomposition of the overall graph likelihood:

$$P(\mathcal{G}|\mathbf{z}) = \prod_{i=1}^{|\mathcal{V}|} P(\mathbf{L}[v_i, :]|\mathbf{L}[v_1, :], ..., \mathbf{L}[v_{i-1}, :], \mathbf{z}). \quad (9.13)$$

When we generate row $L[v_i, :]$, we condition on all the previous generated rows $L[v_j, :]$ with j $< i$.

GraphRNN

- GraphRNN approach use a hierarchical RNN to model the edge dependencies in Equation (9.13).
- The first RNN in the hierarchical model—termed the graph-level RNN—is used to model the state of the graph that has been generated so far. Formally, the graph-level RNN maintains a hidden state h_i, which is updated after generating each row of the adjacency matrix $\boldsymbol{L}[v_i, :]$:

$$\mathbf{h}_{i+1} = \mathrm{RNN}_{\mathrm{graph}}(\mathbf{h}_i, \mathbf{L}[v_i, L]), \qquad (9.14)$$

where we use RNN_{graph} to denote a generic RNN state update with h_i corresponding to the hidden state and $\boldsymbol{L}[v_i, L]$ to the observation.

- The second RNN—termed the node-level RNN or RNN_{node}—generates the entries of $\boldsymbol{L}[v_i, :]$ in an autoregressive manner. RNN_{node} takes the graph-level hidden state h_i as an initial input and then sequentially generates the binary values of $\boldsymbol{L}[v_i, :]$.

GraphRNN

- The overall GraphRNN approach is called hierarchical because the node-level RNN is initialized at each time-step with the current hidden state of the graph-level RNN.

- Note that computing the likelihood in Equation (9.13) requires that we assume a particular ordering over the generated nodes.

$$P(\mathcal{G}|\mathbf{z}) = \prod_{i=1}^{|\mathcal{V}|} P(\mathbf{L}[v_i, :]|\mathbf{L}[v_1, :], ..., \mathbf{L}[v_{i-1}, :], \mathbf{z}). \tag{9.13}$$

- GraphRNN model could, in principle, be used as a decoder or generator within a VAE or GAN framework, respectively.

Limitations

- GraphRNN still generates unrealistic artifacts (e.g., long chains) when trained on samples of grids.
- GraphRNN can be difficult to train and scale to large graphs due to the need to backpropagate through many steps of RNN recurrence (catastrophic forgetting problem).

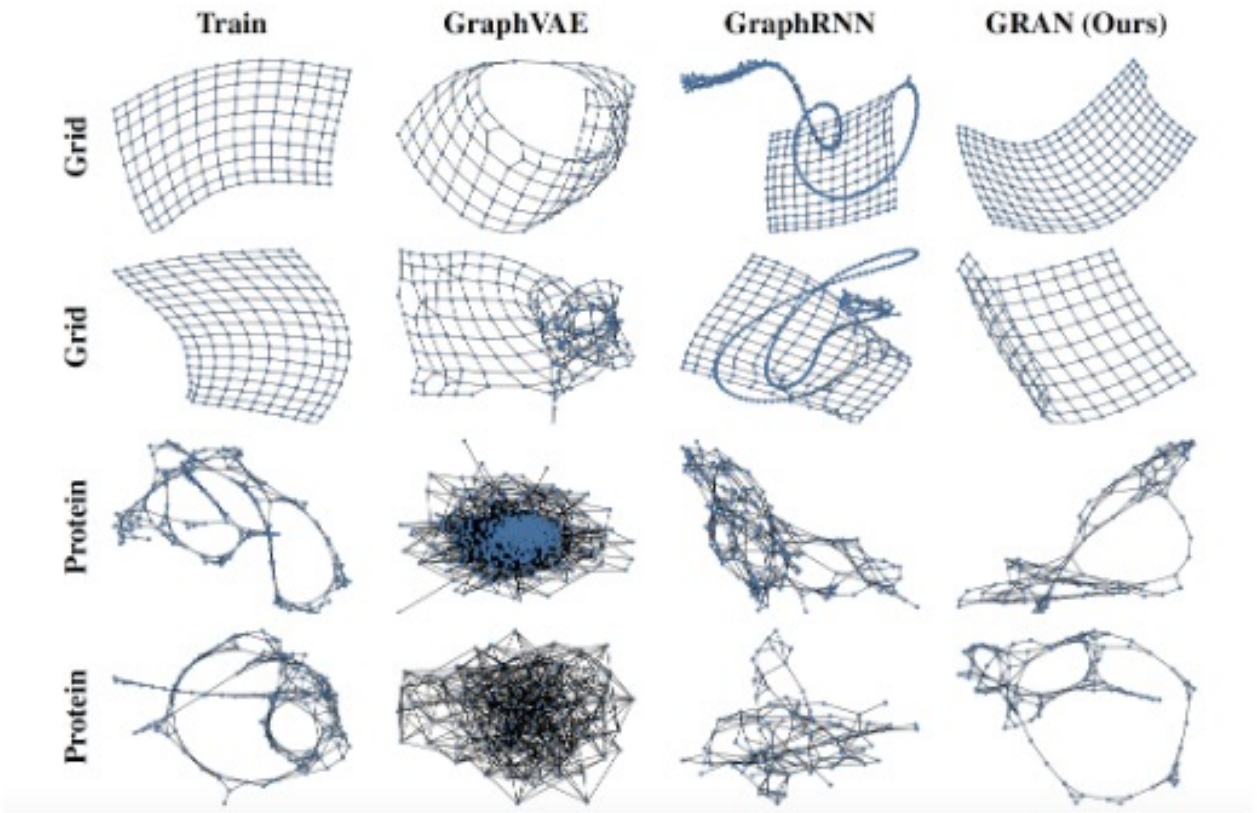


Figure 9.3: Examples of graphs generated by a basic graph-level VAE (Section 9.1), as well as the GraphRNN and GRAN models. Each row corresponds to a different dataset. The first column shows an example of a real graph from the dataset, while the other columns are randomly selected samples of graphs generated by the corresponding model Liao et al., 2019a.

Graph Recurrent Attention Networks (GRAN)

- Instead of using RNNs to model the autoregressive generation process, GRAN uses GNNs. We can model the conditional distribution of each row of the adjacency matrix by running a GNN on the graph that has been generated so far

$$P(\mathbf{L}[v_i,:]|\mathbf{L}[v_1,:],...,\mathbf{L}[v_{i-1},:],\mathbf{z}) \approx \text{GNN}(\mathbf{L}[v_1:v_{i-1},:],\tilde{\mathbf{X}}). \quad (9.15)$$

where $L[v_1 : v_{i-1}, :]$ denotes the lower-triangular adjacency matrix of the graph that has been generated up to generation step i, $\hat{\boldsymbol{X}}$ is the randomly sampled feature matrix.

It does not need to maintain a long and complex history in a graph-level RNN. Instead, the GRAN model explicitly conditions on the already generated graph using a GNN at each generation step.

Evaluating Graph Generation

- Evaluating generative models is a challenging task, as there is no natural notion of accuracy or error (no test set).
- The current practice is to analyze different statistics of the generated graphs, and to compare the distribution of statistics for the generated graphs to a test set.
- Formally, assume we have set of graph statistics $S = (s_1, s_2, \ldots, s_n)$, where each of these statistics $s_{i,G} : R \rightarrow [0,1]$ is assumed to define a univariate distribution over R for a given graph G, e.g., the degree distribution, the distribution of clustering coefficients, and the distribution of different motifs or graphlets. Computed s_i on both a test graph $s_{i,G_{test}}$ and a generated graph $s_{i,G_{gen}}$ and compute the distance

$$d(s_{i,\mathcal{G}_{\text{test}}}, s_{i,\mathcal{G}_{\text{gen}}}) = \sup_{x \in \mathbb{R}} |s_{i,\mathcal{G}_{\text{test}}}(x) - s_{i,\mathcal{G}_{\text{gen}}}(x)|. \qquad (9.16)$$

Example of Graph Generation Evaluation

	Grid				Protein				3D Point Cloud			
	$\lvert V\rvert_{max} = 361$, $\lvert E\rvert_{max} = 684$ $\lvert V\rvert_{avg} \approx 210$, $\lvert E\rvert_{avg} \approx 392$				$\lvert V\rvert_{max} = 500$, $\lvert E\rvert_{max} = 1575$ $\lvert V\rvert_{avg} \approx 258$, $\lvert E\rvert_{avg} \approx 646$				$\lvert V\rvert_{max} = 5037$, $\lvert E\rvert_{max} = 10886$ $\lvert V\rvert_{avg} \approx 1377$, $\lvert E\rvert_{avg} \approx 3074$			
	Deg.	Clus.	Orbit	Spec.	Deg.	Clus.	Orbit	Spec.	Deg.	Clus.	Orbit	Spec.
Erdos-Renyi	0.79	2.00	1.08	0.68	$5.64e^{-2}$	1.00	1.54	$9.13e^{-2}$	0.31	1.22	1.27	$4.26e^{-2}$
GraphVAE*	$7.07e^{-2}$	$7.33e^{-2}$	0.12	$1.44e^{-2}$	0.48	$7.14e^{-2}$	0.74	0.11	-	-	-	-
GraphRNN-S	0.13	$3.73e^{-2}$	0.18	0.19	$4.02e^{-2}$	$\mathbf{4.79e^{-2}}$	0.23	0.21	-	-	-	-
GraphRNN	$1.12e^{-2}$	$\mathbf{7.73e^{-5}}$	$\mathbf{1.03e^{-3}}$	$\mathbf{1.18e^{-2}}$	$1.06e^{-2}$	0.14	0.88	$1.88e^{-2}$	-	-	-	-
GRAN	$\mathbf{8.23e^{-4}}$	$3.79e^{-3}$	$1.59e^{-3}$	$1.62e^{-2}$	$\mathbf{1.98e^{-3}}$	$4.86e^{-2}$	**0.13**	$\mathbf{5.13e^{-3}}$	$\mathbf{1.75e^{-2}}$	**0.51**	**0.21**	$\mathbf{7.45e^{-3}}$

Table 1: Comparison with other graph generative models. For all MMD metrics, the smaller the better. *: our own implementation, -: not applicable due to memory issue, Deg.: degree distribution, Clus.: clustering coefficients, Orbit: the number of 4-node orbits, Spec.: spectrum of graph Laplacian.

Molecule Generation

- Many works within the general area of graph generation are focused specifically on the task of molecule generation.
- The goal of molecule generation is to generate molecular graph structures that are both valid (e.g., chemically stable) and ideally have some desirable properties (e.g., medicinal properties or solubility).
- Unlike the general graph generation problem, research on molecule generation can benefit substantially from domain-specific knowledge for both model design and evaluation strategies.