Group Meeting - September 04, 2020

Paper review & Research progress

Truong Son Hy *

*Department of Computer Science
The University of Chicago

Ryerson Physical Lab

Marcus Aurelius

The **happiness** of your life depends upon the quality of your **thoughts**.

February 2020, The Art Institute of Chicago

Buddha

1. **Meditation brings wisdom**; lack of meditation leaves ignorance. Know well what leads you forward and what hold you back, and choose the path that leads to wisdom.
2. A **monk** is simply a **traveler**, except the **journey** is **inwards**.

Paper

Graph Representation Learning, William L. Hamilton (McGill University, 2020) `https://www.cs.mcgill.ca/~wlh/grl_book/`

- Chapter 8. Traditional graph generation approaches.
- Chapter 9. Deep Generative Models.

Traditional graph generation approaches (1)

Erdös-Rényi (ER) Model:

$$p(\boldsymbol{A}_{u,v} = 1) = r, \qquad \forall u, v \in \mathcal{V}, \qquad u \neq v$$

where $r \in [0, 1]$ is parameter controlling the density of the graph. Edges are generated **independently**.

Traditional graph generation approaches (2)

Stochastic Block Models (SBM):

- We specify a number γ as the number of different blocks $\{C_1, .., C_\gamma\}$.
- Every node $u \in \mathcal{V}$ has a probability p_i of belonging to block i:

$$p_i = p(u \in \mathcal{C}_i), \qquad \sum_{i=1}^{\gamma} p_i = 1$$

- Block-to-block edge probabilities are specified by a matrix $\boldsymbol{C} \in [0, 1]^{\gamma \times \gamma}$ where $C_{i,j}$ is the probability of an edge occuring between a node in block $\mathcal{C}_i$ and a node in block $\mathcal{C}_j$.
- SBM is able to generate graphs that exhibit community structure.

Traditional graph generation approaches (3)

Preferential Attachment (PA) [Albert and Barabási, 2002]:

- **Assumption**: Real-world graphs exhibit **power law** degree distribution. The probability of a node u having degree d_u is:

$$p(d_u = k) \propto k^{-\alpha}, \qquad \alpha > 1$$

Intuitively, the **heavy tailed** power distribution leads to a large number of nodes with small degrees, but a small number of nodes with extremely large degrees.

Traditional graph generation process (4)

Preferential Attachment (PA) [Albert and Barabási, 2002]:

- **Generative process**:
  1. Initialize a fully connected graph with m_0 nodes.
  2. Iteratively add $n - m_0$ nodes to this graph. At iteration t, we add a new node u to the graph, and we connect u to $m < m_0$ existing nodes according to the following distribution:

$$p(\boldsymbol{A}_{u,v} = 1) = \frac{d_v^{(t)}}{\sum_{v' \in \mathcal{V}^{(t)}} d_{v'}^{(t)}}$$

  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.

- The generation process is **autoregressive** (iterative approach). I think it is similar to Dirichlet Process in which high-degree nodes tend to have more connections.

Deep Generative Models

Traditional methods

Limitation:

1. Rely on a fixed, hand-crafted generation process.
2. Lack the ability to learn a generative model from data.

Generative models

Goal: Design models that can observe a set of graphs $\{\mathcal{G}_1, .., \mathcal{G}_n\}$ and learn to generate graphs with similar characteristics as this training set.

- **All-at-once**: VAEs, GANs.
- **Autoregressive** (generate a graph *incrementally*): LSTM-based language models, GRNN, GRANs, etc.

Variational Autoencoder Approaches (1)

Main components:

1. **Probabilistic encoder** $q_\theta(\boldsymbol{Z}|\mathcal{G})$ defines a distribution over *latent representations*. We specify the latent conditional distribution as $\boldsymbol{Z} \sim \mathcal{N}(\mu_\theta(\mathcal{G}), \sigma_\theta(\mathcal{G}))$, where μ_θ and σ_θ are neural networks that generate the mean and variance parameters for a normal distribution (that we sample latent embeddings $\boldsymbol{Z}$ from).
2. **Probabilistic decoder** $p_\theta(\boldsymbol{A}|\boldsymbol{Z})$, from which we can sample realistic graphs (i.e. adjacency matrices) $\hat{\boldsymbol{A}} \sim p_\theta(\boldsymbol{A}|\boldsymbol{Z})$ by conditioning on a latent variable $\boldsymbol{Z}$.
3. **Prior distribution** $p(\boldsymbol{Z})$ over the latent space. Assume $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$.

Variational Autoencoder Approaches (2)

Motivation

Motivated from the theory of variational inference **[Wainwright and Jordan, 2008]** (that I presented 3 weeks ago).

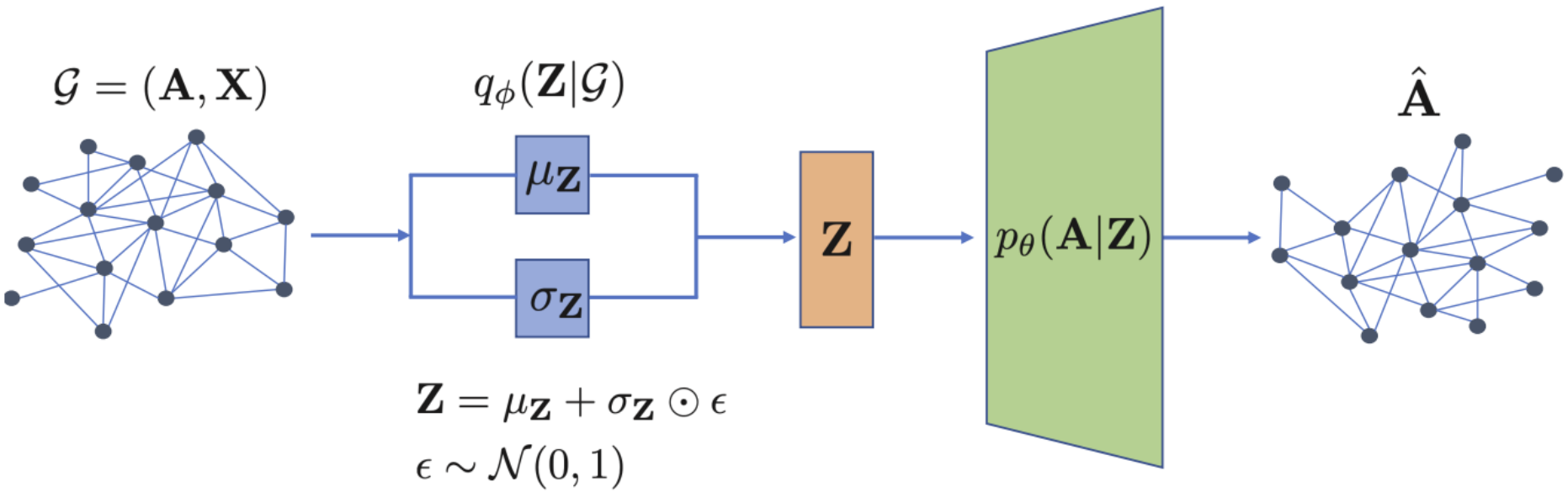
Given a set of training graphs $\{\mathcal{G}_1, .., \mathcal{G}_n\}$, we can train a VAE model by minimizing the evidence likelihood lower bound (ELBO):

$$\mathcal{L} = \sum_{\mathcal{G}_i \in \{\mathcal{G}_1, .., \mathcal{G}_n\}} \mathbb{E}_{q_\theta(\boldsymbol{Z}|\mathcal{G}_i)}[p_\theta(\mathcal{G}_i|\boldsymbol{Z})] - \mathrm{KL}(q_\theta(\boldsymbol{Z}|\mathcal{G}_i)||p(\boldsymbol{Z}))$$

Basic idea:

- Maximize the reconstruction ability of our decoder $\mathbb{E}_{q_\theta(\boldsymbol{Z}|\mathcal{G}_i)}[p_\theta(\mathcal{G}_i|\boldsymbol{Z})]$.
- Minimize the KL-divergence between our posterior latent distribution $q_\theta(\boldsymbol{Z}|\mathcal{G}_i)$ and the prior $p(\boldsymbol{Z})$.

Variational Autoencoder Approaches (3)



Node-level Latents – VGAE (1)

Key idea:

- First proposed by **[Kipf and Welling, 2016]**: Variational Graph Autoencoder (VGAE).
- However, the authors proposed VGAE model as an approach to generate node embeddings, but they did not intend it as a generative model to sample new graphs.
- The encoder generates latent representations for each node in the graph.
- The decoder takes pairs of embeddings as input and uses these embeddings to predict the likelihood of an edge between the two nodes.

Node-level Latents – VGAE (2)

The encoder has two separate GNNs to generate mean and variance parameters, conditioned on the input:

$$\mu_{\boldsymbol{Z}} = \text{GNN}_{\mu}(\boldsymbol{A}, \boldsymbol{X}) \in \mathbb{R}^{|\mathcal{V}| \times d}$$

$$\log \sigma_{\boldsymbol{Z}} = \text{GNN}_{\sigma}(\boldsymbol{A}, \boldsymbol{X}) \in \mathbb{R}^{|\mathcal{V}| \times d}$$

Reparameterization trick to sample a set of latent node embeddings:

$$\boldsymbol{Z} = \epsilon \odot \exp(\log(\sigma_{\boldsymbol{Z}})) + \mu_{\boldsymbol{Z}} \in \mathbb{R}^{|\mathcal{V}| \times d}, \qquad \epsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{1})$$

[Kipf and Welling, 2016] employs a simple dot-product decoder:

$$p_{\theta}(\boldsymbol{A}_{u,v} = 1 | \boldsymbol{z}_u, \boldsymbol{z}_v) = \gamma(\boldsymbol{z}_u^T \boldsymbol{z}_v)$$

$$p_{\theta}(\mathcal{G} | \boldsymbol{Z}) = \prod_{(u,v) \in \mathcal{V}^2} p_{\theta}(\boldsymbol{A}_{u,v} = 1 | \boldsymbol{z}_u, \boldsymbol{z}_v)$$

Graph-level Latents – GraphVAE (1)

Proposed by **Simonovsky and Komodakis, 2018**:

- GraphVAE only has a single graph - level embedding:

$$\boldsymbol{z}_{\mathcal{G}} \sim \mathcal{N}(\mu_{\boldsymbol{z}_{\mathcal{G}}}, \sigma_{\boldsymbol{z}_{\mathcal{G}}})$$

- In contrast, VGAE defines a posterior distributions for each node.

The encoder:

$$\mu_{\boldsymbol{z}_{\mathcal{G}}} = \text{POOL}_{\mu}(\text{GNN}_{\mu}(\boldsymbol{A}, \boldsymbol{X})) \in \mathbb{R}^{|\mathcal{V}| \times d}$$

$$\sigma_{\boldsymbol{z}_{\mathcal{G}}} = \text{POOL}_{\sigma}(\text{GNN}_{\sigma}(\boldsymbol{A}, \boldsymbol{X})) \in \mathbb{R}^{|\mathcal{V}| \times d}$$

where $\text{POOL} : \mathbb{R}^{|\mathcal{V}| \times d} \to \mathbb{R}^{d}$ denotes a pooling function that maps a matrix of node embeddings $\boldsymbol{Z} \in \mathbb{R}^{|\mathcal{V}| \times d}$ to a graph - level embedding vector $\boldsymbol{z}_{\mathcal{G}} \in \mathbb{R}^{d}$.

Graph-level Latents – GraphVAE (2)

With a Bernoulli distributional assumption, the decoder uses an MLP to map the latent vector $\boldsymbol{z}_{\mathcal{G}}$ to a matrix $\hat{\boldsymbol{A}} \in [0,1]^{|\mathcal{V}|\times|\mathcal{V}|}$ of edge probabilities:

$$\hat{\boldsymbol{A}} = \gamma(\text{MLP}(\boldsymbol{z}_{\mathcal{G}}))$$

The posterior distribution:

$$p_\theta(\mathcal{G}|\boldsymbol{z}_{\mathcal{G}}) = \prod_{(u,v)\in\mathcal{V}\times\mathcal{V}} \hat{\boldsymbol{A}}_{u,v}\boldsymbol{A}_{u,v} + (1-\hat{\boldsymbol{A}}_{u,v})(1-\boldsymbol{A}_{u,v})$$

where $\boldsymbol{A}$ denotes the true adjacency, and $\hat{\boldsymbol{A}}$ denotes the predicted edge probabilities.

Graph-level Latents – GraphVAE (3)

Key challenges:

1. Assumption of independent Bernoulli distributions for each edge $\rightarrow$ Graphical models!
2. Assumption of a fixed number of nodes $\rightarrow$ Specify empirical distribution of graph sizes from the training data.
3. We do not know the correct ordering (graph matching):

$$p_\theta(\mathcal{G}|\boldsymbol{z}_\mathcal{G}) = \max_{\pi \in \Pi} \prod_{(u,v) \in \mathcal{V} \times \mathcal{V}} \hat{\boldsymbol{A}}^\pi_{u,v} \boldsymbol{A}_{u,v} + (1 - \hat{\boldsymbol{A}}^\pi_{u,v})(1 - \boldsymbol{A}_{u,v})$$

Specify a particular ordering function π:

$$p_\theta(\mathcal{G}|\boldsymbol{z}_\mathcal{G}) \approx \prod_{(u,v) \in \mathcal{V} \times \mathcal{V}} \hat{\boldsymbol{A}}^\pi_{u,v} \boldsymbol{A}_{u,v} + (1 - \hat{\boldsymbol{A}}^\pi_{u,v})(1 - \boldsymbol{A}_{u,v})$$

or a bit smarter:

$$p_\theta(\mathcal{G}|\boldsymbol{z}_\mathcal{G}) \approx \sum_{\pi_i \in \{\pi_1, .., \pi_n\}} \prod_{(u,v) \in \mathcal{V} \times \mathcal{V}} \hat{\boldsymbol{A}}^{\pi_i}_{u,v} \boldsymbol{A}_{u,v} + (1 - \hat{\boldsymbol{A}}^{\pi_i}_{u,v})(1 - \boldsymbol{A}_{u,v})$$

Adversarial Approaches (1)

Basic idea of GAN-based generative models:

- Define a trainable **generator network** $g_\theta : \mathbb{R}^d \to \mathcal{X}$ that takes a random seed $\boldsymbol{z} \in \mathbb{R}^d$ as input, and generates realistic (but fake) data samples $\hat{\boldsymbol{x}} \in \mathcal{X}$.
- Define a **discriminator network** $d_\phi : \mathcal{X} \to [0, 1]$ that outputs the probability a given input is fake (distinguishing between real data samples $\boldsymbol{x} \in \mathcal{X}$, and samples generated by the generator $\hat{\boldsymbol{x}} \in \mathcal{X}$).

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

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

where $p_{\text{data}}(\boldsymbol{x})$ denotes the empirical distribution of real data samples, and $p_{\text{seed}}(\boldsymbol{z})$ denotes the random seed distribution (e.g. standard multivariate normal).

Adversarial Approaches (2)

[De Cao and Kipf, 2018] (less successful than VAE-based methods) The generator generates a matrix of edge probabilities given a seed vector $\boldsymbol{z}$:

$$\hat{\boldsymbol{A}} = \gamma(\text{MLP}(\boldsymbol{z}))$$

Given this matrix of edge probabilities, we generate a discrete adjacency matrix $\boldsymbol{A} \in \mathbb{Z}^{|\mathcal{V}|\times|\mathcal{V}|}$ by sampling independent Bernoulli variables for each edge $\boldsymbol{A}_{u,v} \sim \text{Bernoulli}(\hat{\boldsymbol{A}}_{u,v})$. The discriminator is any GNN-based graph classification model.

Advantage and Disadvantage:

1. GAN-based models do not require any node ordering (vs. VAE-based models).
2. Training minimax optimization is difficult.

Autoregressive Models (1)

To model edge dependencies, we define the likelihood of a graph given a latent representation $\boldsymbol{z}$ by decomposing the overall likelihood into a set of independent edge likelihoods:

$$p(\mathcal{G}|\boldsymbol{z}) = \prod_{(u,v)\in\mathcal{V}\times\mathcal{V}} p(\boldsymbol{A}_{u,v}|\boldsymbol{z})$$

In 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:

$$p(\mathcal{G}|\boldsymbol{z}) = \prod_{i=1}^{|\mathcal{V}|} p(\boldsymbol{L}_{v_i}|\boldsymbol{L}_{v_1}, .., \boldsymbol{L}_{v_{i-1}}, \boldsymbol{z})$$

where $\boldsymbol{L}$ denotes the lower-triangle portion of the adjacency matrix $\boldsymbol{A}$.

Autoregressive Models (2)

[You et al., 2018] GraphRNN:

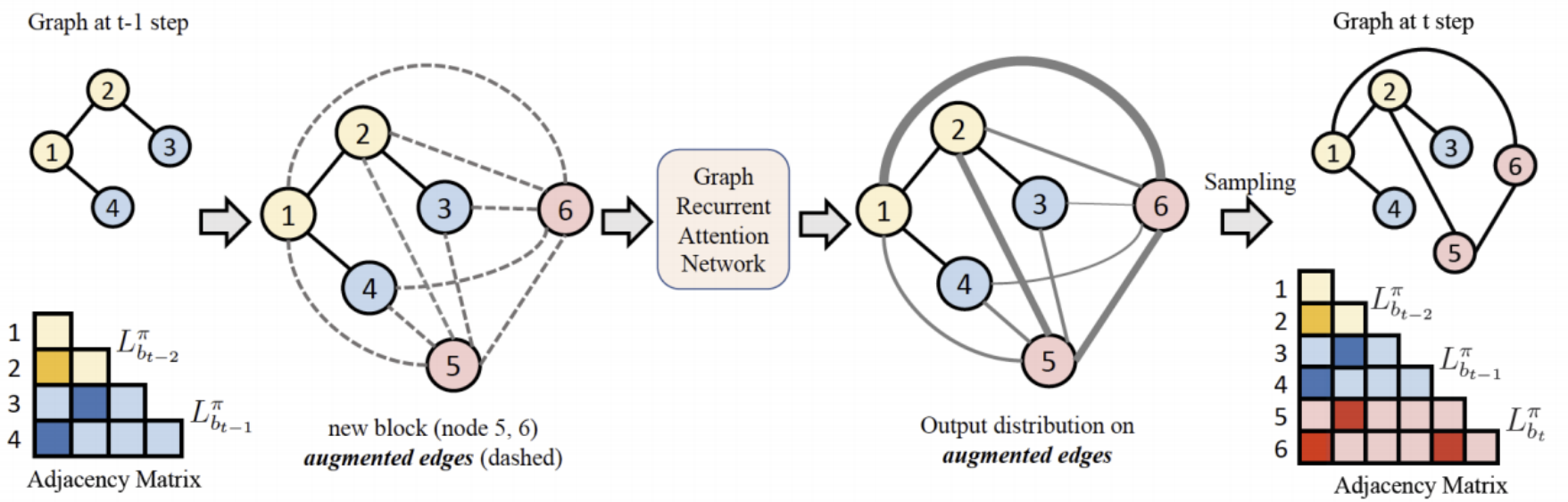
1. Graph-level RNN maintains a hidden state $\boldsymbol{h}_i$, which is updated after generating each row of the adjacency matrix $\boldsymbol{L}_{v_i}$:

$$\boldsymbol{h}_{i+1} = \text{RNN}_{\text{graph}}(\boldsymbol{h}_i, \boldsymbol{L}_{v_i})$$

2. Node-level RNN generates the entries of $\boldsymbol{L}_{v_i}$ in an autoregressive manner. RNN_{node} takes the graph-level hidden state $\boldsymbol{h}_i$ as an initial input, and then sequentially generates the binary values of $\boldsymbol{L}_{v_i}$ assuming a conditional Bernoulli distribution for each entry.

Autoregressive Models (3)

[Lieo et al., 2019] GRAN (graph recurrent attention networks) models the conditional distribution of each row of the adjacency matrix by running a GNN on the graph that has been generated so far.



Evaluating graph generation (1)

[Liao et al., 2019] 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.

Assume we have a set of graph statistics $\mathcal{S} = (s_1, s_2, .., s_n)$ where each of these statistics $s_{i,\mathcal{G}} : \mathbb{R} \to [0, 1]$ is assumed to define a univariate distribution over $\mathbb{R}$ for a given graph $\mathcal{G}$. For example:

- Degree distribution
- Distribution of clustering coefficients
- Distribution of different motifs or graphlets

Evaluating graph generation (2)

Given a particular statistic s_i, total variance distance betwen the statistics of a test graph $s_{i,\mathcal{G}_{test}}$ and a generated graph $s_{i,\mathcal{G}_{gen}}$:

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

To measure the performance, we compute the average pairwise distributional distance between a set of generated graphs and graphs in a test set (e.g. Wasserstein distance).

Molecule generation (special case):

- Graph structures need to be both valid (e.g. chemically stable), and ideally have some desirable properties (e.g. medicinal properties, or solubility).
- Unlike general graph generation, molecule generation can benefit substantially from domain-specific knowledge for both model design and evaluation strategies.

Evaluating graph generation (3)