GraphRNN: A Deep Generative Model for Graphs (24 Feb 2018)

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, Jure Leskovec
Presented by: Jesse Bettencourt and Harris Chan
March 9, 2018

University of Toronto, Vector Institute
Introduction: Generative Model for Graphs

Modeling graphs is fundamental for studying networks e.g. medical, chemical, social

Goal:
Model and efficiently sample complex distributions over graphs
Learn generative model from observed set of graphs
Challenges in Graph Generation

**Large and variable output spaces**
Graph with \( n \) nodes requires \( n^2 \) to fully specify structure
Number of nodes and edges varies between different graphs

**Non-unique representations**
Distributions over graphs without assuming fixed set of nodes
\( n \) node graph represented by up to \( n! \) equivalent adjacency matrices
\( \pi \in \Pi \) is arbitrary node ordering

**Complex, non-local dependencies**
New edges depend on previously generated edges
Overview to GraphRNN

Decompose graph generation into two RNNs:

- **Graph-level**: generates sequence of nodes
- **Edge-level**: generates sequence of edges for each new node
Graph $G \sim p(G)$ with $n$ nodes under node ordering $\pi$
Define mapping $f_S$ from $G$ to sequence
$$S^\pi = f_S(G, \pi) = (S_1^\pi, \ldots, S_n^\pi)$$ (1)
Each sequence element is adjacency vector
$$S_i^\pi \in \{0, 1\}^{i-1} \quad i \in \{1, \ldots, n\}$$
for edges between node $\pi(v_i)$ and $\pi(v_j)$, $j \in \{1, \ldots, i - 1\}$
Modeling Graphs as Sequences

Graph $G \sim p(G)$ with $n$ nodes under node ordering $\pi$
Define mapping $f_S$ from $G$ to sequence

$$S^{\pi} = f_S(G, \pi) = (S_1^{\pi}, \ldots, S_n^{\pi})$$

(1)

Each sequence element is adjacency vector

$$S_i^{\pi} \in \{0, 1\}^{i-1} \quad i \in \{1, \ldots, n\}$$

for edges between node $\pi(v_i)$ and $\pi(v_j)$, $j \in \{1, \ldots, i - 1\}$
Graph $G \sim p(G)$ with $n$ nodes under node ordering $\pi$
Define mapping $f_s$ from $G$ to sequence
\[
S^\pi = f_s(G, \pi) = (S^\pi_1, \ldots, S^\pi_n)
\] (1)
Each sequence element is adjacency vector
\[
S^\pi_i \in \{0, 1\}^{i-1} \quad i \in \{1, \ldots, n\}
\]
for edges between node $\pi(v_i)$ and $\pi(v_j), j \in \{1, \ldots, i - 1\}$
Modeling Graphs as Sequences

Graph $G \sim p(G)$ with $n$ nodes under node ordering $\pi$
Define mapping $f_s$ from $G$ to sequence

$$S^\pi = f_s(G, \pi) = (S_1^\pi, \ldots, S_n^\pi) \quad (1)$$

Each sequence element is adjacency vector

$$S_i^\pi \in \{0, 1\}^{i-1} \quad i \in \{1, \ldots, n\}$$

for edges between node $\pi(v_i)$ and $\pi(v_j)$, $j \in \{1, \ldots, i - 1\}$
Modeling Graphs as Sequences

Graph $G \sim p(G)$ with $n$ nodes under node ordering $\pi$

Define mapping $f_{S}$ from $G$ to sequence

$$S^\pi = f_{S}(G, \pi) = (S^\pi_1, \ldots, S^\pi_n)$$

(1)

Each sequence element is adjacency vector

$$S^\pi_i \in \{0, 1\}^{i-1} \quad i \in \{1, \ldots, n\}$$

for edges between node $\pi(v_i)$ and $\pi(v_j)$, $j \in \{1, \ldots, i - 1\}$
Instead of learning $p(G)$ sample, $\pi \sim \Pi$ to get observations of $S^\pi$

Then learn $p(S^\pi)$ modeled autoregressively:

$$p(G) = \sum_{S^\pi} p(S^\pi) 1[f_G(S^\pi) = G] \quad (3)$$

Exploiting sequential structure of $S^\pi$, decompose $p(S^\pi)$

$$P(S^\pi) = \prod_{i=1}^{n+1} p(S^\pi_i | S^\pi_1, \ldots, S^\pi_{i-1}) \quad (4)$$

$$= \prod_{i=1}^{n+1} p(S^\pi_i | S^\pi_{<i})$$
Motivating GraphRNN

Model $p(G)$
Distribution over graphs

$\downarrow$

Model $p(S^\pi)$
Distribution over sequence of edge connections

$\downarrow$

Model $p(S^\pi_i | S^\pi_{<i})$
Distribution over edge connections for $i$-th node conditioned on previous nodes’ edge connections

parameterize with an expressive neural network
GraphRNN Framework

**Idea:** Use an RNN that consists of a *state-transition* function and an *output* function:

\[
    h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi) \quad (5)
\]
\[
    \theta_i = f_{\text{out}}(h_i) \quad (6)
\]

- \( h_i \in \mathbb{R}^d \) encodes the state of the graph generated so far
- \( S_{i-1}^\pi \) encodes adjacency for most recently generated node \( i-1 \)
- \( \theta_i \) specifies the distribution of next node’s adjacency vector

\[
    S_i^\pi \sim \mathcal{P}_{\theta_i}
\]

- \( f_{\text{trans}} \) and \( f_{\text{out}} \) can be arbitrary neural networks
- \( \mathcal{P}_{\theta_i} \) can be an arbitrary distribution over binary vectors
**Idea:** Use an RNN that consists of a *state-transition* function and an *output* function:

\[
\begin{align*}
    h_i &= f_{\text{trans}}(h_{i-1}, S_i^{\pi}) \\
    \theta_{i+1} &= f_{\text{out}}(h_i)
\end{align*}
\] (5) (6)

- \( h_i \in \mathbb{R}^d \) encodes the state of the graph generated so far
- \( S_i^{\pi} \) encodes adjacency for most recently generated node \( i \)
- \( \theta_{i+1} \) specifies the distribution of next node’s adjacency vector

\[
S_{i+1}^{\pi} \sim \mathcal{P}_{\theta_{i+1}}
\]

- \( f_{\text{trans}} \) and \( f_{\text{out}} \) can be arbitrary neural networks
- \( \mathcal{P}_{\theta_i} \) can be an arbitrary distribution over binary vectors.
Idea: Use an RNN that consists of a state-transition function and an output function:

\[ h_i = f_{\text{trans}}(h_{i-1}, S^\pi_i) \]  \hspace{1cm} (5)

\[ \theta_{i+1} = f_{\text{out}}(h_i) \] \hspace{1cm} (6)

\[ S^\pi_{i+1} \sim P_{\theta_{i+1}} \]

GraphRNN Framework Corrected
**Algorithm 1** GraphRNN inference algorithm

**Input:** RNN-based transition module $f_{\text{trans}}$, output module $f_{\text{out}}$, probability distribution $\mathcal{P}_{\theta_i}$ parameterized by $\theta_i$, start token $\text{SOS}$, end token $\text{EOS}$, empty graph state $h'$

**Output:** Graph sequence $S^\pi$

$S_0^\pi = \text{SOS}$, $h_0 = h'$, $i = 0$

repeat

$i = i + 1$

$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$ \{update graph state\}

$\theta_i = f_{\text{out}}(h_i)$

$S_i^\pi \sim \mathcal{P}_{\theta_i}$ \{sample node $i$’s edge connections\}

until $S_i^\pi$ is $\text{EOS}$

Return $S^\pi = (S_1^\pi, ..., S_i^\pi)$
Algorithm 1 GraphRNN inference algorithm

Input: RNN-based transition module \( f_{\text{trans}} \), output module \( f_{\text{out}} \), probability distribution \( P_{\theta_i} \) parameterized by \( \theta_i \), start token \( \text{SOS} \), end token \( \text{EOS} \), empty graph state \( h' \)

Output: Graph sequence \( S^\pi \)

\[
S^\pi_0 = \text{SOS}, \ h_0 = h', \ i = 0
\]

repeat

\[
i = i + 1
\]

\[
h_i = f_{\text{trans}}(h_{i-1}, S^\pi_{i-1}) \quad \{\text{update graph state}\}
\]

\[
\theta_{i+1} = f_{\text{out}}(h_i)
\]

\[
S^\pi_i \sim P_{\theta_{i+1}} \quad \{\text{sample node } i+1's \text{ edge connections}\}
\]

until \( S^\pi_{i+1} \) is EOS

Return \( S^\pi = (S^\pi_1, ..., S^\pi_i) \)
Objective: $\prod p_{model}(S^{\pi})$ over all observed graph sequences

Implement $f_{\text{trans}}$ as **Gated Recurrent Unit (GRU)**

But different assumptions about $p(S_{i}^{\pi}|S_{<i}^{\pi})$ for each variant:

1. **Multivariate Bernoulli** (GraphRNN-S):
   - $f_{\text{out}}$ is a MLP with sigmoid activation that outputs $\theta_{i+1} \in \mathbb{R}^{i}$
   - $\theta_{i+1}$ parameterizes the multivariate Bernoulli
   - $S_{i+1}^{\pi} \sim \mathcal{P}_{\theta_{i+1}}$ independently
GraphRNN Variants

**Objective:** \( \prod p_{model}(S^\pi) \) over all observed graph sequences

Implement \( f_{\text{trans}} \) as **Gated Recurrent Unit (GRU)**

But different assumptions about \( p(S^\pi_i|S^\pi_{<i}) \) for each variant:

2. **Dependent Bernoulli sequence** (GraphRNN):

\[
p(S^\pi_i|S^\pi_{<i}) = \prod_{j=1}^{i-1} p(S^\pi_{i,j}|S^\pi_{i,<j}, S^\pi_{<i}) \quad (7)
\]

- \( S^\pi_{i,j} \in \{0, 1\} \) indicating if node \( \pi(v_i) \) is connected to node \( \pi(v_j) \)
- \( f_{\text{out}} \) is a *edge-level* RNN generates the edges of a given node
Idea: Apply BFS ordering to the graph $G$ with node permutation $\pi$ before generating the sequence $S^\pi$

Benefits:

- **Reduce overall # of seq to consider**
  Only need to train on all possible BFS orderings, rather than all possible node permutations

- **Reduce the number of edge predictions**
  Edge-level RNN only predicts $M$ edges, the maximum size of the BFS queue
BFS Order Leads To Fixed Size $S_i^\pi$

$S_i^\pi \in \mathbb{R}^M$ represents “sliding window” over nodes in the BFS queue. 
Zero-pad all $S_i^\pi$ to be a length $M$ vector:

$$S_i^\pi = (A_{\text{max}(1,i-M),i}^{\pi}, \ldots, A_{i-1,i}^{\pi})^T, \ i \in \{2, \ldots, n\}$$  \hspace{1cm} (9)
Experiments
### Datasets

3 Synthetic and 2 real graph datasets:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th># Graphs</th>
<th>Graph Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community</td>
<td>Synthetic</td>
<td>500</td>
<td>$60 \leq |V| \leq 160$</td>
<td>2-community, Erdős-Rényi model (E-R)</td>
</tr>
<tr>
<td>Grid</td>
<td>Synthetic</td>
<td>100</td>
<td>$100 \leq</td>
<td>V</td>
</tr>
<tr>
<td>B-A</td>
<td>Synthetic</td>
<td>500</td>
<td>$100 \leq</td>
<td>V</td>
</tr>
<tr>
<td>Protein</td>
<td>Real</td>
<td>918</td>
<td>$100 \leq</td>
<td>V</td>
</tr>
<tr>
<td>Ego</td>
<td>Real</td>
<td>757</td>
<td>$50 \leq</td>
<td>V</td>
</tr>
</tbody>
</table>
Baseline Methods & Settings

- Compared GraphRNN to traditional models and deep learning baselines:

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>Erdős-Rényi Model (E-R) (Erdős &amp; Rényi, 1959)</td>
</tr>
<tr>
<td></td>
<td>Barabási-Albert Model (B-A) (Albert &amp; Barabási, 2002)</td>
</tr>
<tr>
<td></td>
<td>Kronecker graph models (Leskovec et al., 2010)</td>
</tr>
<tr>
<td></td>
<td>Mixed-membership stochastic block models (MMSB) (Airoldi et al., 2008)</td>
</tr>
<tr>
<td>Deep learning</td>
<td>GraphVAE (Simonovsky &amp; Komodakis, 2018)</td>
</tr>
<tr>
<td></td>
<td>DeepGMG (Li et al., 2018)</td>
</tr>
</tbody>
</table>

- 80%-20% train-test split
- All models trained with early stopping
- Traditional methods: learn from a single graph, so train a separate model for each training graph in order to compare with these methods
- Deep learning baselines: use smaller dataset:
  - Community-small: $12 \leq |V| \leq 20$
  - Ego-small: $4 \leq \|V\| \leq 18$
Evaluating Generated Graph Via MMD Metric

Existing:

- Visual Inspection
- Simple comparisons of average statistics between the two sets

Proposed:
A metric based on **Maximum Mean Discrepancy (MMD)**, to compare all moments of their empirical distributions using an exponential kernel with Wasserstein distance.
Figure 2: Visualization of graphs from grid dataset (Left group), community dataset (Middle group) and Ego dataset (Right group). Within each group, graphs from training set (First row), graphs generated by GraphRNN(Second row) and graphs generated by Kronecker, MMSB and B-A baselines respectively (Third row) are shown. Different visualization layouts are used for different datasets.
Comparison with traditional models

Table 1: Comparison of GraphRNN to traditional graph generative models using MMD. \(\max(|V|), \max(|E|)\) of each dataset is shown.

<table>
<thead>
<tr>
<th></th>
<th>Community (160,1945)</th>
<th>Ego (399,1071)</th>
<th>Grid (361,684)</th>
<th>Protein (500,1575)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-R</td>
<td>0.021</td>
<td>1.243</td>
<td>0.049</td>
<td>0.508</td>
</tr>
<tr>
<td>B-A</td>
<td>0.268</td>
<td>0.322</td>
<td>0.047</td>
<td>0.275</td>
</tr>
<tr>
<td>Kronecker</td>
<td>0.259</td>
<td>1.685</td>
<td>0.069</td>
<td>0.108</td>
</tr>
<tr>
<td>MMSB</td>
<td>0.166</td>
<td>1.59</td>
<td>0.054</td>
<td>0.304</td>
</tr>
<tr>
<td>GraphRNN-S</td>
<td>0.055</td>
<td>0.016</td>
<td>0.041</td>
<td>0.090</td>
</tr>
<tr>
<td>GraphRNN</td>
<td><strong>0.014</strong></td>
<td><strong>0.002</strong></td>
<td><strong>0.039</strong></td>
<td><strong>0.077</strong></td>
</tr>
</tbody>
</table>

- GraphRNN had **80% decrease of MMD** on average compared with traditional baselines
- GraphRNN-S performed well on Protein: may not involve highly complex edge dependencies
### Table 2: GraphRNN compared to state-of-the-art deep graph generative models on small graph datasets using MMD and negative log-likelihood (NLL). $(\max(|V|), \max(|E|))$ of each dataset is shown. (DeepVAE and GraphVAE cannot scale to the graphs in Table 1.)

<table>
<thead>
<tr>
<th></th>
<th>Community-small (20,83)</th>
<th>Ego-small (18,69)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Degree</td>
<td>Clustering</td>
</tr>
<tr>
<td>GraphVAE</td>
<td>0.35</td>
<td>0.98</td>
</tr>
<tr>
<td>DeepGMG</td>
<td>0.22</td>
<td>0.95</td>
</tr>
<tr>
<td>GraphRNN-S</td>
<td><strong>0.02</strong></td>
<td>0.15</td>
</tr>
<tr>
<td>GraphRNN</td>
<td>0.03</td>
<td><strong>0.03</strong></td>
</tr>
</tbody>
</table>

- GraphRNN had **90% decrease of MMD** on average compared with deep learning baselines
- 22% smaller average NLL gap compared to other deep models
Experiments: Evaluation with Graph Statistics

Figure 3: Average degree (Left) and clustering coefficient (Right) distributions of graphs from test set and graphs generated by GraphRNN and baseline models.

- GraphRNN generated graphs’ average statistics closely matches the overall test set distribution.
Experiments: Robustness

Interpolate between (B-A) and (E-R) graphs
Randomly perturb \([0\%, 20\%, \ldots, 100\%]\) edges of B-A graphs
0\% (B-A) \rightleftharpoons 100\% (E-R)

**Figure 4:** MMD performance of different approaches on degree (Left) and clustering coefficient (Right) under different noise level.

GraphRNN maintains strong performance as we interpolate between these structures, indicating high robustness and versatility.