Efficient Graph Generation with Graph Recurrent Attention Networks

Renjie Liao1,2,3, Yujia Li4, Yang Song5, Shenlong Wang1,2,3, Charlie Nash4, William L. Hamilton2,3, David Duvenaud1,3, Raquel Urtasun1,2,3, Richard Zemel1,3,8

University of Toronto1, Uber ATG Toronto2, Vector Institute3, DeepMind4, Stanford University5, McGill University6, Mila - Quebec Artificial Intelligence Institute7, Canadian Institute for Advanced Research8

Generative Model of Graphs
Model the distribution of graph \( G = (V, E) \):

\[
P(G) = \sum_{\pi} P(L, \pi),
\]

where \( L \in \{ \text{binary adjacency matrix} \} \) and \( \pi \in \{ \text{node ordering} \} \).

Contributions

- Our approach consists of \( O(N) \) auto-regressive generation steps, where a block of nodes and associated edges are generated per step.
- We propose an attention-based GNN that better utilizes the topology of the already generated graph, reduces the dependency on the node ordering, and distinguishes multiple newly added nodes.
- We capture the correlation between multiple generated edges via a mixture of Bernoulli output distribution per step.
- We approximate the likelihood by marginalizing output distribution per step.

Graph Recurrent Attention Networks (GRAN)

Auto-regressive Decomposition:

\[
p(L^t, \pi) = \prod_{i=1}^{t} P(L_i^{t-1}|L_0^t, \cdots, L_{i-1}^t, \pi)
\]

Initial Node Representation:

\[
h_b^i = \text{WEL}_b^i + b, \quad \forall i \in t
\]

Message Passing:

\[
a_j' = \text{Sigmoid}(g(k_j, h_j)),
\]

\[
h_j' = \text{GRU}(k_j', \sum_{i \in N(j)} a_i'm_i).
\]

Output Distribution:

\[
P(L_i^{t-1}|L_0^t, \cdots, L_{i-1}^t, \pi) = \sum_{k \in \pi} \alpha_k \text{MLP}_a(h_b^i - h_b^k)
\]

Approximated Likelihood:

\[
P(G) = \sum_{\pi} P(L^t, \pi) \geq \sum_{\pi \in Q} P(L^t, \pi) \quad \text{e.g.} \quad Q = \{ \text{BFS, TDFS, } \pi \text{-degree descent, } \pi \text{-k-core, default} \}
\]

Experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>( V_{\text{train}} )</th>
<th>( E_{\text{train}} )</th>
<th>( V_{\text{val}} )</th>
<th>( E_{\text{val}} )</th>
<th>( V_{\text{test}} )</th>
<th>( E_{\text{test}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
<td>210 392 361 664</td>
<td>( 10^3 )</td>
<td>278 364 258</td>
<td>9.65e-3</td>
<td>258 364 258</td>
<td>9.65e-3</td>
</tr>
<tr>
<td>Protein</td>
<td>258 646 500 1575</td>
<td>( 10^3 )</td>
<td>258 364 258</td>
<td>9.65e-3</td>
<td>258 364 258</td>
<td>9.65e-3</td>
</tr>
<tr>
<td>3D Point Cloud</td>
<td>1377 3074 5037 10886</td>
<td>( 10^3 )</td>
<td>258 364 258</td>
<td>9.65e-3</td>
<td>258 364 258</td>
<td>9.65e-3</td>
</tr>
</tbody>
</table>

Table: For all MMD metrics, the smaller the better. *: our own implementation, -: not applicable due to memory issue, Deg.: degree of node orbits, Spec.: spectrum of graph Laplacian.

Figure: Efficiency vs. sample quality. The bar and line plots are the MMD implementation, -: not applicable due to memory issue, Deg.: degree of node orbits, Spec.: spectrum of graph Laplacian.

Visualization

(a) Train GraphVAE
(b) GRAN

Figure: Train and sampled graphs on the protein dataset.

(c) GRAN

Figure: Train and sampled graphs on the 3D point cloud dataset.

(d) GRAN

Figure: Train and sampled graphs on the grid and protein datasets.

Code (Pytorch): https://github.com/1rjc0san/GRAN

Reference:


