Appendix for "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models"

Jiaxuan You [*1] **Rex Ying** [*1] **Xiang Ren** [2] **William L. Hamilton** [1] **Jure Leskovec** [1]

1. Appendix

1.1. Implementation Details of GraphRNN

In this section we detail parameter setting, data preparation and training strategies for GraphRNN.

We use two sets of model parameters for GraphRNN. One larger model is used to train and test on the larger datasets that are used to compare with traditional methods. One smaller model is used to train and test on datasets with nodes up to 20. This model is only used to compare with the two most recent preliminary deep generative models for graphs proposed in (Li et al., 2018; Simonovsky & Komodakis, 2018).

For GraphRNN, the graph-level RNN uses 4 layers of GRU cells, with 128 dimensional hidden state for the larger model, and 64 dimensional hidden state for the smaller model in each layer. The edge-level RNN uses 4 layers of GRU cells, with 16 dimensional hidden state for both the larger model and the smaller model. To output the adjacency vector prediction, the edge-level RNN first maps the highest layer of the 16 dimensional hidden state to a 8 dimensional vector through a MLP with ReLU activation, then another MLP maps the vector to a scalar with sigmoid activation. The edge-level RNN is initialized by the output of the graph-level RNN at the start of generating S_i^π, $\forall 1 \leq i \leq n$. Specifically, the highest layer hidden state of the graph-level RNN is used to initialize the lowest layer of edge-level RNN, with a liner layer to match the dimensionality. During training time, teacher forcing is used for both graph-level and edge-level RNNs, i.e., we use the groud truth rather than the model's own prediction during training. At inference time, the model uses its own preditions at each time steps to generate a graph.

[*]Equal contribution [1]Department of Computer Science, Stanford University, Stanford, CA, 94305 [2]Department of Computer Science, University of Southern California, Los Angeles, CA, 90007. Correspondence to: Jiaxuan You <jiaxuan@stanford.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

For the simple version GraphRNN-S, a two-layer MLP with ReLU and sigmoid activations respectively is used to generate S_i^π, with 64 dimensional hidden state for the larger model, and 32 dimensional hidden state for the smaller model. In practice, we find that the performance of the model is relatively stable with respect to these hyperparameters.

We generate the graph sequences used for training the model following the procedure in Section 2.3.4. Specifically, we first randomly sample a graph from the training set, then randomly permute the node ordering of the graph. We then do the deterministic BFS discussed in Section 2.3.4 over the graph with random node ordering, resulting a graph with BFS node ordering. An exception is in the robustness section, where we use the node ordering that generates B-A graphs to get graph sequences, in order to see if GraphRNN can capture the underlying preferential attachment properties of B-A graphs.

With the proposed BFS node ordering, we can reduce the maximum dimension M of S_i^π, illustrated in Figure 1. To set the maximum dimension M of S_i^π, we use the following empirical procedure. We randomly ran 100000 times the above data pre-processing procedure to get graph with BFS node orderings. We remove the all consecutive zeros in all resulting S_i^π, to find the empirical distribution of the dimensionality of S_i^π. We set M to be roughly the 99.9 percentile, to account for the majority dimensionality of S_i^π. In principle, we find that graphs with regular structures tend to have smaller M, while random graphs or community graphs tend to have larger M. Specifically, for community dataset, we set $M = 100$; for grid dataset, we set $M = 40$; for B-A dataset, we set $M = 130$; for protein dataset, we set $M = 230$; for ego dataset, we set $M = 250$; for all small graph datasets, we set $M = 20$.

The Adam Optimizer is used for minibatch training. Each minibatch contains 32 graph sequences. We train the model for 96000 batchs in all experiments. We set the learning rate to be 0.001, which is decayed by 0.3 at step 12800 and 32000 in all experiments.

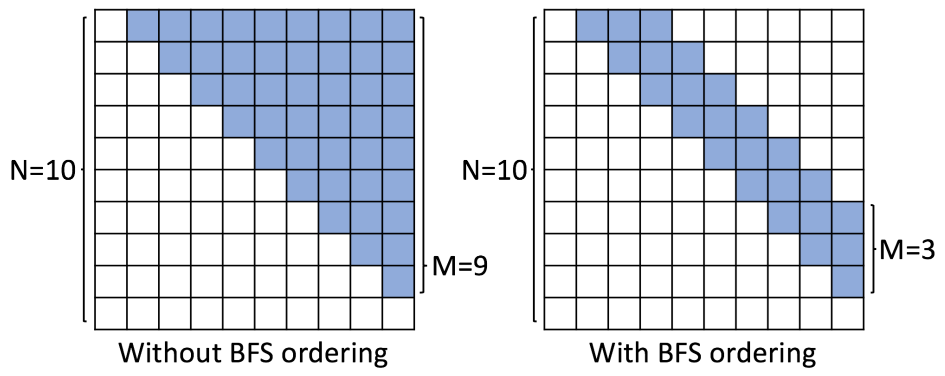

Figure 1. Illustrative example of reducing the maximum dimension M of S_i^π through the BFS node ordering. Here we show the adjacency matrix of a graph with $N = 10$ nodes. Without the BFS node ordering (Left), we have to set $M = N - 1$ to encode all the necessary connection information (shown in dark square). With the BFS node ordering, we could set M to be a constant smaller than N (we show $M = 3$ in the figure).

1.2. Running Time of GraphRNN

Training is performed on only 1 Titan X GPU. For the protein dataset that consists of about 1000 graphs, each containing about 500 nodes, training converges at around 64000 iterations. The runtime is around 12 to 24 hours. This also includes pre-processing, batching and BFS, which are currently implemented using CPU without multi-threading. The less expressive GraphRNN-S variant is about twice faster. At inference time, for the above dataset, generating a graph using the trained model only takes about 1 second.

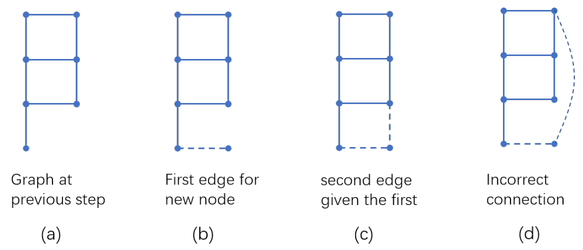

Figure 2. Illustration that generation of ladder networks relies on dependencies modeled by GraphRNN.

1.3. More Details on GraphRNN's Expressiveness

We illustrate the intuition underlying the good performance of GraphRNNon graphs with regular structures, such as grid and ladder networks. Figure 2 (a) shows the generation process of a ladder graph at an intermediate step. At this time step, the ground truth data (under BFS node ordering) specifies that the new node added to the graph should make an edge to the node with degree 1. Note that node degree is a function of $S^\pi_{<i}$, thus could be approximated by a neural network.

Once the first edge has been generated, the new node should make an edge with another node of degree 2. However, there are multiple ways to do so, but only one of them gives a valid grid structure, *i.e.* one that forms a 4-cycle with the new edge. GraphRNN crucially relies on the edge-level RNN and the knowledge of the previously added edge, in order to distinguish between the correct and incorrect connections in Figure 2 (c) and (d).

1.4. Code Overview

In the code repository, `main.py` consists of the main training pipeline, which loads datasets and performs training and inference. It also consists of the `Args` class, which stores the hyper-parameter settings of the model. `model.py` consists of the RNN, MLP and loss function modules that are use to build GraphRNN. `data.py` contains the minibatch sampler, which samples a random BFS ordering of a batch of randomly selected graphs. `evaluate.py` contains the code for evaluating the generated graphs using the MMD metric introduced in Sec. 4.3.

Baselines including the Erdős-Rényi model, Barabási-Albert model, MMSB, and rge very recent deep generative models (GraphVAE, DeepGMG) are also implemented in the `baselines` folders. We adopt the C++ Kronecker graph model implementation in the SNAP package [1].

1.5. Proofs

1.5.1. Proof of Proposition 1

We use the following observation:

Observation 1. *By definition of BFS, if $i < k$, then the children of v_i in the BFS ordering come before the children*

[1]The SNAP package is available at `http://snap.stanford.edu/snap/index.html`.

of v_k that do not connect to $v_{i'}$, $\forall 1 \leq i' \leq i$.

By definition of BFS, all neighbors of a node v_i include the parent of v_i in the BFS tree, all children of v_i which have consecutive indices, and some children of $v_{i'}$ which connect to both $v_{i'}$ and v_i, for some $1 \leq i' \leq i$. Hence if $(v_i, v_{j-1}) \in E$ but $(v_i, v_j) \notin E$, v_{j-1} is the last children of v_i in the BFS ordering. Hence $(v_i, v_{j'}) \notin E, \forall j \leq j' \leq n$.

For all $i' \in [i]$, supposed that $(v_{i'}, v_{j'-1}) \in E$ but $(v_{i'}, v_{j'}) \notin E$. By Observation 1, $j' < j$. By conclusion in the previous paragraph, $(v_{i'}, v_{j''}) \notin E, \forall j' \leq j'' \leq n$. Specifically, $(v_{i'}, v_{j''}) \notin E, \forall j \leq j'' \leq n$. This is true for all $i' \in [i]$. Hence we prove that $(v_{i'}, v_{j'}) \notin E, \forall 1 \leq i' \leq i$ and $j \leq j' < n$.

1.5.2. Proof of Proposition 2

As proven in Kolouri et al. 2016, this Wasserstein distance based kernel is a positive definite (p.d.) kernel. By properties that linear combinations, product and limit (if exists) of p.d. kernels are p.d. kernels, $k_W(p, q)$ is also a p.d. kernel.[2] By the Moore-Aronszajn theorem, a symmetric p.d. kernel induces a unique RKHS. Therefore Equation (9) holds if we set k to be k_W.

1.6. Extension to Graphs with Node and Edge Features

Our GraphRNN model can also be applied to graphs where nodes and edges have feature vectors associated with them. In this extended setting, under node ordering π, a graph G is associated with its node feature matrix $X^\pi \in \mathbb{R}^{n \times m}$ and edge feature matrix $F^\pi \in \mathbb{R}^{n \times k}$, where m and k are the feature dimensions for node and edge respectively. In this case, we can extend the definition of S^π to include feature vectors of corresponding nodes as well as edges $S_i^\pi = (X_i^\pi, F_i^\pi)$. We can adapt the f_{out} module, by using a MLP to generate X_i^π and an edge-level RNN to genearte F_i^π respectively. Note also that directed graphs can be viewed as an undirected graphs with two edge types, which is a special case under the above extension.

1.7. Extension to Graphs with Four Communities

To further show the ability of GraphRNN to learn from community graphs, we further conduct experiments on a four-community synthetic graph dataset. Specifically, the data set consists of 500 four community graphs with $48 \leq |V| \leq 68$. Each community is generated by the Erdős-Rényi model (E-R) (Erdős & Rényi, 1959) with $n \in [|V|/4 - 2, |V|/4+2]$ nodes and $p = 0.7$. We then add $0.01|V|^2$ inter-community edges with uniform probability. FIgure 3 shows the comparison of visualization of generated graph using GraphRNN and other baselines. We observe that in contrast to baselines, GraphRNN consistently generate 4-community graphs and each community has similar structure to that in the training set.

[2]This can be seen by expressing the kernel function using Taylor expansion.

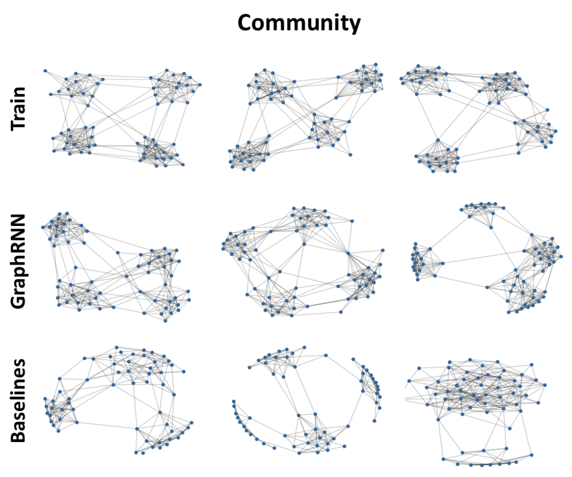

Figure 3. Visualization of graph dataset with four communities. Graphs from training set (First row), graphs generated by GraphRNN(Second row) and graphs generated by Kronecker, MMSB and B-A baselines respectively (Third row) are shown.

References

Erdős, P. and Rényi, A. On random graphs I. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

Kolouri, S., Zou, Y., and Rohde, G. Sliced Wasserstein kernels for probability distributions. In *CVPR*, 2016.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs, 2018. URL https://openreview.net/forum?id=Hy1d-ebAb.

Simonovsky, M. and Komodakis, N. GraphVAE: Towards generation of small graphs using variational autoencoders, 2018. URL https://openreview.net/forum?id=SJlhPMWAW.