Autoregressive Generative Modeling of Weighted Graphs

Richard Williams
University of California, Los Angeles
rlwilliams34@ucla.edu

Eric Nalisnick
Johns Hopkins University
nalisnick@jhu.edu

Andrew Holbrook
University of California, Los Angeles
aholbroo@ucla.edu

Abstract

Weighted graphs are ubiquitous throughout biology, chemistry, and the social sciences. However, most current deep generative models are designed for unweighted graphs and are not easily extended to weighted topologies. This paper proposes two autoregressive models on weighted graphs: Adj-LSTM and BiGG-E. Experiments on a variety of benchmark datasets demonstrate that both models adequately capture distributions over weighted graphs while remaining computationally scalable. Specifically, we experiment with Erdős–Rényi, tree, lobster, and 3D point cloud graph data sets.

1 Introduction

Graph generative modeling is a prominent field in machine learning due to its application in a variety of domains, such as molecule generation, semantic parsing, and phylogenetic tree construction (Guo and Zhao, 2020). Most graph generative models, however, focus on graph topology that precludes the learning of a joint distribution over edges and weights. Indeed, such a model must account for complex, non-local dependencies between the graph's edges and corresponding weights, where the sampling of a weight depends on the existence of an edge that further complicates this distribution.

In this work, we compare three autoregressive models that learn distributions over weighted graphs: Adjacency-LSTM (Adj-LSTM), BiGG-E, and BiGG+GCN. Adj-LSTM is a fully expressive but slower model that directly parameterizes a graph's adjacency matrix; BiGG-E is an extension of the existing BiGG model (Dai et al., 2020) that takes advantage of graph sparsity to jointly generate a weighted graph in log-linear time; and BiGG+GCN is a two-stage model consisting of the original BiGG model and a graph convolutional network (GCN) that generates edge weights conditioned on an unweighted graph. Experiments on benchmark datasets indicate the models are capable of learning joint distributions over weighted graphs and highlight relative computational performance.

2 Background

Data. We define a weighted graph $G = (V, E, W)$ as a graph with a set of nodes $V = \{v_1, ..., v_n\}$, edges $E \subseteq V \times V = \{(v_i, v_j) | v_i, v_j \in V\}$, and corresponding edge weights $W : V \times V \to \mathbb{R}^+$, where $W(v_i, v_j) = w_{ij}$ if $(v_i, v_j) \in E$ and otherwise is 0. A graph under node ordering π is represented by its weighted adjacency matrix $A^\pi \in \mathbb{R}^{n \times n}$ with entries $W(v_i, v_j)$, from which the probability of observing a graph G is expressed as $p(G) = p(|V| = n) \sum_\pi p(A^\pi)$ (Dai et al., 2020).

Given that the probability of observing a particular graph necessitates summing over all possible node orderings, we assume a single canonical ordering π of G as in Liao et al. (2019) to estimate the lower bound $p(G) \simeq p(|V| = n)p(A^{\pi(G)})$ (Liao et al., 2019). As implemented in Dai et al. (2020), we estimate $p(|V| = n)$ as a simple multinomial distribution over the number of nodes in the training graph set and learn an expressive model for $p(A^{\pi(G)})$ using deep autoregressive neural networks.

R. Williams et al., Autoregressive Generative Modeling of Weighted Graphs (Extended Abstract). Presented at the Third Learning on Graphs Conference (LoG 2024), Virtual Event, November 26–29, 2024.

Previously Proposed Generative Models for Graphs. Numerous generative models have successfully learned distributions over graph topologies with varying degrees of scalability. Early work consists of VAE-based methods (Grover et al., 2019; Kipf and Welling, 2016), autoregressive models (Dai et al., 2020; Liao et al., 2019; You et al., 2018), and more recently, score-based diffusion models (Jo et al., 2022; Niu et al., 2020). Most implementations do not directly model a joint distribution over continuous non-negative edge weights. Such weighted graphs are a common occurrence in the sciences, which motivates us to build general models tailored to these graphs.

BiGG (Dai et al., 2020) addresses scalability issues that arise in modeling large graphs. The original model leverages the sparsity inherent to many real-world graphs to directly generate the edge-set in an autoregressive manner by directly estimating

$$p(A) = \prod_{i=1}^{m} p(e_i | \{e_{k;k<i}\}) \tag{1}$$

A row i of the adjacency matrix is constructed by building a decision tree that recursively divides the interval of possible node connections $[v_1, v_{i-1}]$ in half through a sequence of Bernoulli decisions corresponding to whether an edge connection exists in the left-half or the right-half of the node interval. Once an interval $[v_j, v_j]$ is reached in the decision tree (denoted as a leaf of the tree), an edge between nodes v_i and v_j is formed. As autoregressive models scale well with larger graphs, we use these models for weighted graph generation.

3 Methods and Contributions

3.1 Joint Modeling of Topology and Edge Weights

We use the framework established in Section 2 to modify (1) as follows: given an edge exists between nodes v_i and v_j, sample a corresponding weight w from a conditional density function $p(w)$. Since no weight is sampled for non-edge connections, the probability of observing an adjacency matrix can be expressed as

$$p(A) = \prod_{i=1}^{m} p(e_i, w_i | \{(w_k, e_k)\}_{k<i}) = \prod_{i=1}^{m} p(e_i | \{(w_k, e_k)\}) p(w_i | e_i, \{(w_k, e_k)\}). \tag{2}$$

Using (2), we can compute our objective function as the log likelihood over the edges of the adjacency matrix as

$$\mathcal{L}(\theta; E, W) = \log p_\theta(E) p_\theta(W|E) = \sum_{i=1}^{m} \log p_\theta(e_i, \cdot) + \sum_{i=1}^{m} \log p_\theta(w_i | e_i, \cdot) \tag{3}$$

Last, we place a prior distribution on the non-negative weights w_i using a normal random variable $\epsilon_i \sim N(\mu_i, \sigma_i)$ transformed with the softplus function $\text{Softplus}(\epsilon_i) = \log(1 + \exp(\epsilon_i))$. In our studies, such a transformation of a normal random variable performs best with gradient-based optimization by providing enough flexibility in modeling distributions, where work such as Rodríguez and Dunson (2011) demonstrates that a probit transformation of a random normal variable provided a prior capable of generating a rich class of distributions. Thus, with the softplus-normal prior placed on the weights, the term $\log p(w_i | e_i, \cdot)$ in (3) becomes

$$\log p_\theta(w_i | e_i, \cdot) = -\frac{1}{2} \log(\sigma_i^2) - \frac{1}{2\sigma_i^2} (\log(e^{w_i} - 1) - \mu_i)^2$$

where the models use multilayer perceptrons (MLPs) to parameterize μ_i and σ_i^2.

3.2 Proposed Models

Adjacency-LSTM. Our first model directly generates the lower half of A row-wise in $O(n^2)$ time by using an LSTM that models topology and weights jointly (see Appendix 6.2 for details). The objective function to be minimized is augmented from (2) as the model directly generates A (Appendix 6.3)

$$\mathcal{L}(\theta; E, W) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} (1 - e_{ij}) \log(1 - p_{ij}) + e_{ij} \log p_{ij} + e_{ij} \log p(w_{ij} | e_{ij})$$

BiGG-E. We extend the BiGG model to generate weighted sparse graphs with n nodes and m edges in $O((n+m)\log n)$ time (Dai et al., 2020). Weight generation in Section 3.1 is implemented by sampling a weight once a leaf in the decision tree is reached. Such an extension allows for the joint modeling of edge existences and corresponding weights in an autoregressive manner: the generation of future edges and weights depends on the set of edges and weights generated in the graph thus far. The objective function is given by (3), as we are directly generating the edge-set of A.

BiGG+GCN. We developed a two-stage comparison model that (1) uses the original BiGG to generate an unweighted graph and (2) uses a GCN conditioned on the topology of the unweighted graph to sample the edge weights. This model serves as a comparison model to the joint estimation of edge weights and topologies with BiGG-E versus independent estimation.

4 Experiment

Our experiments answer the following questions: (1) do the models autoregressively generate weights; (2) do the models capture complex weighted graph distributions; and (3) do the models scale computationally to larger graphs? We use the same evaluation protocol per Liao et al. (2019). We use measures on graph topologies and weights independently and jointly. Our measures of graph topology use the maximum mean discrepancy (MMD) (Gretton et al., 2012) statistic with test functions on degree distribution, clustering coefficient, and spectrum of the normalized unweighted Laplacian. Our measures of graph weights include first and second-order summary statistics on the edge weights, as well as the MMD statistic on the distribution of weights per graph. Our measure that captures graph topology and edge weights jointly is the MMD on the spectrum of the normalized weighted Laplacian matrix. Finally, we use an Erdos-Renyi baseline model. Results are in Table 1, with complete summary measures located in Appendix 6.6.

Data. We use the following datasets to evaluate each models' generative quality (Appendix 6.4).

- **Erdős–Rényi**: 100 graphs that represent a null case to test whether the models are capturing the distribution of weighted graphs under an Erdős–Rényi model. (Erdős and Rényi, 1959)
- **Tree**: 1000 graphs of bifurcating trees. The tree weights are constructed to test for the autoregressiveness in each model: weights globally have a standard deviation of 2, but weights per tree only have a standard deviation of 1. Hence, we should observe half as much variability among weights when stratified among trees than when pooled together, on average.
- **3D Point Cloud**: Graphs are of 41 household objects (Neumann et al., 2013).
- **Lobster**: 1000 graphs where edges at most two hops away from the backbone.

(1) Weight Generation and Autoregressiveness. We hypothesized both Adj-LSTM and BiGG-E would outperform BiGG+GCN and the baseline model with respect to weight generation. While Table 1 indicates the baseline models performed well when weights were generated independently as in the Erdős–Rényi and lobster data sets, both struggled to capture the dependence found in the weights for the tree data set, as evidenced by the lack in difference between the global weight standard deviation (s_w) and the per-tree standard deviation (s_T). On the other hand, the Adj-LSTM and BiGG-E both demonstrated a clear difference in s_w (1.955 and 1.962, respectively) and s_T (1.118 and 1.084, respectively).

(2) Capturing Distributions. For the lobster data set, we observed that both Adj-LSTM and BiGG-E outperformed BiGG+GCN and the baseline model on all observed metrics. The superior topological spectral and degree MMD measures indicated Adj-LSTM better captured the topological properties of the graphs, whereas BiGG-E provided better weight generation from the superior weighted spectral and weight MMD measures, though the differences appeared minor between the two models. A similar observation occurred for the tree data sets, where both models outperformed the baseline Erdos-Renyi model and maintained similar performance across measures. Overall, all proposed models seemed to be capturing the distributions of smaller graphs quite well.

Table 1: Model performance on datasets. The MMD metrics use the test functions from {Deg., Clus., Spec. using Unweighted (T) or Weighted (W) Laplacian}. For the MMD metrics, smaller values are better. OOM indicates out of memory. Node and edge counts are given as max (avg).

Datasets		Methods			
		Adj-LSTM	BiGG-E	BiGG+GCN	Erdos-Renyi
Erdős–Rényi	Spec-T	0.516	$2.81e^{-3}$	$4.26e^{-3}$	$\mathbf{2.06e^{-3}}$
	Spec-W	0.469	$3.64e^{-3}$	$4.64e^{-3}$	$\mathbf{2.48e^{-3}}$
$\lvert V\rvert_{max} = 749$ (499)	Deg.	0.437	$4.00e^{-3}$	$7.13e^{-3}$	$\mathbf{2.80e^{-3}}$
$\lvert E\rvert_{max} = 2846$ (1349)	Clus.	0.744	0.034	**0.014**	0.015
Tree	Spec-T	$1.67e^{-3}$	$1.45e^{-3}$	$\mathbf{8.55e^{-4}}$	0.268
	Spec-W	$\mathbf{1.13e^{-3}}$	$1.20e^{-3}$	$2.09e^{-3}$	0.109
$\lvert V\rvert = 199, \lvert E\rvert = 198$	s_w	1.955	1.962	1.955	1.994
$\sigma_w = 2, \sigma_T = 1$	s_T	1.118	**1.084**	1.956	1.993
3D Point Cloud	Spec-T	OOM	$1.64e^{-2}$	$\mathbf{8.25e^{-3}}$	0.089
	Spec-W	OOM	$1.73e^{-2}$	$\mathbf{9.50e^{-3}}$	0.112
$\lvert V\rvert_{max} = 5037$ (1377)	Deg.	OOM	0.247	**0.070**	0.418
$\lvert E\rvert_{max} = 10886$ (3074)	MMDWT	OOM	$\mathbf{2.34e^{-5}}$	$3.43e^{-3}$	$9.01e^{-3}$
Lobster	Spec-T	$\mathbf{9.98e^{-4}}$	$1.86e^{-3}$	$1.11e^{-3}$	0.192
	Spec-W	$8.51e^{-4}$	$\mathbf{7.86e^{-4}}$	$1.32e^{-3}$	0.270
$\lvert V\rvert_{max} = 100$ (55)	Deg.	$\mathbf{2.46e^{-4}}$	$8.28e^{-4}$	$4.84e^{-4}$	0.178
$\lvert E\rvert_{max} = 99$ (54)	MMDWt	$4.93e^{-3}$	$\mathbf{3.27e^{-3}}$	$5.09e^{-3}$	0.010

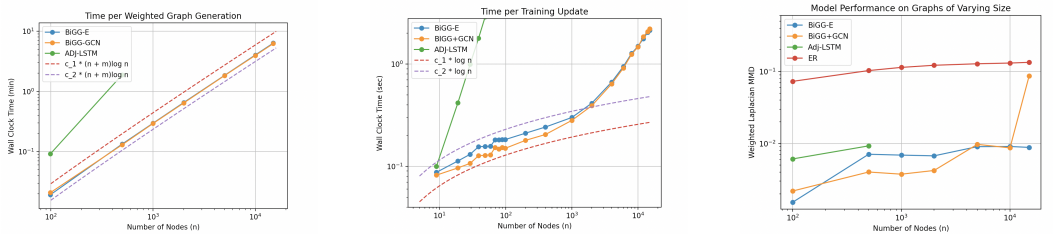

Figure 1: Model Performance on Graph Size

(3) Scalability. We hypothesized that Adj-LSTM would struggle with scalability despite producing high quality smaller graphs. On the other hand, we expected BiGG-E and BiGG+GCN to perform well modeling graphs of any size. Indeed, Table 1 indicates Adj-LSTM performed well on the smaller tree and lobster graphs well, but struggled to scale to the Erdős–Rényi graphs and was computationally infeasible for the 3D Point Clouds. BiGG-E, however, maintained high generative quality with respect to the Erdős–Rényi graphs. BiGG+GCN outperformed BiGG-E on the 3D Point Clouds, suggesting difficulties in jointly training over topologies and weights with respect to large graphs. Figure 1A shows that both models are capable of still sampling weighted graphs in $O((n+m)\log n)$ time, and Figure 1D shows both models train in $O(\log n)$ time. Finally, Figure 1C shows that BiGG-E scales more easily with larger graphs, where BiGG+GCN scaled with graphs up to 10K nodes but begins failing to scale on graphs with 15K nodes.

5 Conclusion and Future Work

We introduced two autoregressive models that can learn complex joint distributions over graphs with edge weights. Both Adj-LSTM and BiGG-E are able to learn from distributions from smaller graphs, and BiGG-E scales to graphs with thousands of nodes. Hence, future work consists of further exploring the benefits of joint modeling edge weights and topologies by learning joint distributions over topologies and vectors of edge and node attributes, and learning conditional distributions over these given node- or edge-related data.

Acknowledgements

We would like to extend our thanks to Dr. Hanjun Dai for his help in extending the code-base of BiGG to incporate edge and node features.

Richard Williams and Andrew Holbrook are supported by NSF grant DMS 2236854.

References

R. Albert and A.-L. Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47, 2002. URL http://link.aps.org/abstract/RMP/v74/p47. 7

H. Dai, A. Nazi, Y. Li, B. Dai, and D. Schuurmans. Scalable deep generative modeling for sparse graphs. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 2302–2312. PMLR, 2020. URL http://dblp.uni-trier.de/db/conf/icml/icml2020.html#DaiNLDS20. 1, 2, 3, 7

P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae (Debrecen)*, 6:290, 1959. URL /brokenurl#snap.stanford.edu/class/cs224w-readings/erdos60random.pdf. 3, 7

A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *J. Mach. Learn. Res.*, 13(1):723–773, Mar. 2012. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2503308.2188410. 3

A. Grover, A. Zweig, and S. Ermon. Graphite: Iterative generative modeling of graphs. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2434–2444. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/grover19a.html. 2, 7

X. Guo and L. Zhao. A systematic survey on deep generative models for graph generation. *arXiv preprint arXiv:2007.06686*, 2020. 1

J. Ingraham, V. Garg, R. Barzilay, and T. Jaakkola. Generative models for graph-based protein design. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/f3a4ff4839c56a5f460c88cce3666a2b-Paper.pdf. 7

J. Jo, S. Lee, and S. J. Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations, 2022. URL https://arxiv.org/abs/2202.02514. 2, 7, 8

M. Karami. Higen: Hierarchical graph generative networks, 2023. URL https://arxiv.org/abs/2305.19337. 7

W. Kawai, Y. Mukuta, and T. Harada. GRAM: scalable generative models for graphs with graph attention mechanism. *CoRR*, abs/1906.01861, 2019. URL http://arxiv.org/abs/1906.01861. 7

T. N. Kipf and M. Welling. Variational Graph Auto-Encoders. In *NIPS Workshop on Bayesian Deep Learning*, BDL 2016, 2016. URL http://bayesiandeeplearning.org/2016/papers/BDL_16.pdf. 2, 7

Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. W. Battaglia. Learning deep generative models of graphs. *CoRR*, abs/1803.03324, 2018. URL http://arxiv.org/abs/1803.03324. 7

R. Liao, Y. Li, Y. Song, S. Wang, W. L. Hamilton, D. Duvenaud, R. Urtasun, and R. S. Zemel. Efficient graph generation with graph recurrent attention networks. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. B. Fox, and R. Garnett, editors, *NeurIPS*, pages 4257–4267, 2019. URL http://dblp.uni-trier.de/db/conf/nips/nips2019.html#LiaoLSWHDUZ19. 1, 2, 3, 7, 8

M. Neumann, P. Moreno, L. Antanas, R. Garnett, and K. Kersting. Graph Kernels for Object Category Prediction in Task-Dependent Robot Grasping. In *Proceedings of the Eleventh Workshop on Mining and Learning with Graphs (MLG–2013)*, Chicago, US, 2013. 3

C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon. Permutation invariant graph generation via score-based generative modeling, 2020. URL https://arxiv.org/abs/2003.00638. 2, 7

A. Rodríguez and D. Dunson. Nonparametric bayesian models through probit stick-breaking processes. *Bayesian Anal.*, 2011. [Accessed 11-09-2024]. 2

H. Sak, A. W. Senior, and F. Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014. URL http://arxiv.org/abs/1402.1128. 9

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, page 5998–6008. Curran Associates, Inc., 2017. URL https://papers.nips.cc/paper/7181-attention-is-all-you-need. 8

J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. Graphrnn: A deep generative model for graphs. *CoRR*, abs/1802.08773, 2018. URL http://dblp.uni-trier.de/db/journals/corr/corr1802.html#abs-1802-08773. 2, 7, 8, 9

6 Appendix

6.1 More Details on Related Work

Graph Generative Models. The earliest graph generative models date back to Erdos-Renyi graphs, which assume an identical and independent probability of existence for all edges in the graph (Erdős and Rényi, 1959). Barbasari and Albert introduce a generative model that uses a preferential attachment mechanism to produce graphs obeying a power law over its degree distribution, where the degree of a node is the number of edges in the graph connected to that node (Albert and Barabasi, 2002). While additional improvements beyond Erdos-Renyi and Barbasari-Albert graphs encapsulate deeper mathematical properties of graphs, such hand-engineered models often fail to capture subtle yet complex dependencies between edges in real-world graphs, motivating the development of expressive deep graph generative models capable of learning potentially nonlinear relationships between the edges of a graph.

Deep generative modeling has proven to be a successful avenue in the learning of probability distributions over graphs, with the deployment of a diverse set of architectures. Early work consists of VAE-based methods (Kipf and Welling, 2016) that require prohibitively costly graph matching, limiting their success on small graphs. Grover et al. (2019) improves upon the VAE framework by parameterizing the encoder-decoder pipeline with graph neural networks that incorporate node and edge features. While Graphite scales to larger graphs compared with earlier models, these gains on larger graphs are limited to downstream representation tasks such as link prediction and node classification. Recently, new score-based diffusion models such as those of Niu et al. (2020) and Jo et al. (2022) have been successfully deployed, where these models attempt to learn from the score function over adjacency matrices of the graphs and apply annealed Langevin sampling dynamics to recover the adjacency matrix. While such models consider weighted adjacency matrices in their generative processes, their use of graph neural networks limits scalability to larger graphs. Further, Langevin sampling dynamics compound this issue by making sampling too slow for large graphs (Jo et al., 2022). Finally, autoregressive models decompose the problem of graph generation into a sequential problem, wherein nodes are added to a graph one by one in a way that subsequent connections depend on previously generated connections (Li et al., 2018; Liao et al., 2019; You et al., 2018). Such models, however, are currently limited to the generation of unweighted graphs, where most prominent examples do not incorporate edge weights.

Modeling Graphs with Edge Weights. Most prior work on graph generative modeling only evaluates the generative quality of graphs with respect to topology, conditioning on edge and node features at most. Even among models that do incporate edge features, most do not formally define a joint distribution over topology and weights, nor do they evaluate whether sampled features match the distribution from which they arise. Furthermore, existing popular autoregressive models such as GraphRNN (You et al., 2018), GRAN (Liao et al., 2019), and BiGG (Dai et al., 2020) only learn from unweighted graphs.

Generative Models on Weighted Graphs. Current work on graph generative models do not directly consider modeling a joint distribution over graph edges and corresponding edge weights. While various models incorporate edge and node features in the graph generative process, such features are either categorical (Kawai et al., 2019; Kipf and Welling, 2016; Li et al., 2018) or are designed to handle a specific class of graphs such as protein graphs (Ingraham et al., 2019). Prior work on autoregressive models focus such as You et al. (2018), Liao et al. (2019), and Dai et al. (2020) exclusively on unweighted graphs, motivating the need to provide an autoregressive model capable of learning over weighted graphs.

Of the models that do learn from weighted graphs, most implementations do not provide a solid foundation for the modeling of continuous non-negative edge weights. Graphite proposes modeling weighted adjacency matrices by parameterizing a simple Gaussian random variable using graph neural networks, but such a distributional assumption introduces the possibility of infeasible negative weights. While Niu et al. (2020) proposes a score-based generative model that incorporates weighted graphs, their model relies on thresholding to produce a weighted adjacency matrix. Further, Niu et al. (2020) only evaluates their performance on discretized adjacency matrices. Finally, a recently proposed hierarchical model HighGen (Karami, 2023) directly models edge weights by assuming a multinomial distribution on the weight. However, such weights may only represent count variables, whereas our weight generation is focused on continuous attributes that can be made general. Our

model addresses these differences by formally defining a joint distribution over topology and edge weights and evaluating the generative performance of our models on the edge weights themselves.

Scalability. Scaling generative models to larger graphs on the order of thousands of nodes remains an ongoing challenge. The primary issue is that most generative models leverage the entire adjacency matrix of a graph, which grows quadratically in size with respect to the number of nodes n in the graph. Further, the VAE-based and score-based diffusion models discussed here utilize graph neural networks, which - although powerful tools when working with graph data - necessitate performing convolutions over the entire adjacency matrix of the graph and thus precludes scaling these models to larger graphs. While such models have been able to scale up to graphs with a few hundred nodes (Jo et al., 2022), we are interested in scaling graphs to thousands of nodes.

Autoregressive models have currently achieved the most progress with respect to scalability. The earliest model GraphRNN uses a BFS-ordering scheme to reduce model complexity of generating a full adjacency matrix, but still scales on the order of $O(n^2)$ with respect to graph size (You et al., 2018). GRAN improves upon the scalability of autoregressive models by using graph neural networks with an attention mechanism to generate blocks of nodes of the graph at a time, but trades this gain in scalability for sample quality as the model must now use a mixture of Bernoulli distributions to estimate edges per block of nodes (Liao et al., 2019).

6.2 Adjacency-LSTM Algorithm

Algorithm 1 outlines the sampling process for a graph of size n using Adj-LSTM. The functions f_p, f_μ, f_σ, and f_w are each parameterized by MLPs with one hidden layer. The function $\text{Pos}(\cdot)$ represents positional encodings from (Vaswani et al., 2017) on the initial row state of each node. The embedding of edge e_{ij} and weight w_{ij} is composed of four components: the first is an embedding of edge existence; the second is an embedding of the weight using an MLP; and the third and fourth are trainable positional embeddings of the current adjacency entry of row i and column j, respectively.

Algorithm 1 Adjacency-LSTM Sampling Algorithm

Input: Number of nodes n
Initialization Initial row node state $s^R_{0,0} = (h_{0,0}, c_{0,0})$
for $i = 1, ..., n$ **do**
  $s_{i0} = s_{i-1,i-1} + \text{Pos}(n + 1 - i)$ {initialize new row node state}
  for $j < i$ **do**
    $s_{ij} = \text{Cat}(s^R_{i,j-1}, s^C_{i-1,j})$ {concatenate the previous node states of row i and column j}
    $p_{ij} = \sigma(f_p(h_{ij}))$
    Sample edge $e_{ij} \sim \text{Bernoulli}(p_{ij})$
    if edge exists **then**
      $\mu_{ij} = f_\mu(h_{ij})$
      $\log \sigma^2_{ij} = f_\sigma(h_{ij})$
      $\epsilon_{ij} \sim \text{Normal}(\mu_{ij}, \sigma_{ij})$
      $w_{ij} = \log(1 + \exp(\epsilon_{ij}))$
    else
      $w_{ij} = 0$
    end if
    $\text{embed}(e_{ij}, w_{ij}) = \text{Cat}(E_{ij}, f_w(w_{ij}), N_i, N_j)$
    $s^*_{ij} = \text{LSTM}(\text{embed}(e_{ij}, w_{ij}); s_{ij})$ {update the adjacency state with edge embedding}
    $s^R_{ij}, s^C_{ij} = \text{Split}(s^*_{ij})$
  end for
  $s^R_{i,i} = s^R_{i,i-1}$ {set final row node state for subsequent row generation}
end for
Output: G with $V = \{1, 2, ..., n\}$ and $E = \{e_{ij}, w_{ij}\}^n_{i=1;j>i}$

6.2.1 Adjacency-LSTM Motivation

The primary issue with using an LSTM to build the adjacency matrix of a graph is that most recurrent neural networks are best suited for linear data. Flattening an adjacency matrix and using an LSTM

is one potential avenue for building a model, but suffers from significant drawbacks – the flattened vector varies based upon the node ordering π, and the model sacrifices the underlying structure of A. (You et al., 2018) Generative models such as GraphRNN and GRAN, which use recurrent neural networks to build generative models of graphs, circumvent this issue by using node-level and graph-level recurrent networks that maintain edge generation and the global structure of the graph, respectively. Adj-LSTM was inspired by such methods and instead uses a partitioning of the hidden state to take advantage of the grid structure of the adjacency matrix directly. We will also show that partitioning the hidden state of a single LSTM provides greater generative quality, as this facilitates information passing between the states of the row and column nodes.

The classic LSTM Cell uses four gates – the input, forget, cell, and output gates – to model dependencies between observations. (Sak et al., 2014) Each gate can be summarized as a linear transformation between the current input x_t and prior hidden state h_{t-1}

$$Z_t = W_i.x_t + W_h.h_{t-1} + b. \tag{4}$$

followed by a sigmoid activation function. As we are now generating a two-dimensional matrix, we re-index (4) as $Z_{ij,t} = W_i.x_{ij} + W_h.h_{ij,t-1} + b.$ to correspond with updating the LSTM with adjacency entry x_{ij}.

To adapt the LSTM architecture to suit a two-dimensional adjacency matrix, we partition the hidden state of the LSTM into $h_{ij} = \begin{bmatrix} h^R_{i,j-1} \\ h^C_{i-1,j} \end{bmatrix}$, where $h^R_{i,j-1}$ and $h^C_{i-1,j}$ are the prior hidden states of the row and column nodes corresponding to entry A_{ij}. As all state updates are the same regardless of which entry A_{ij} is being generated, we drop the subscripts i and j moving forward.

We re-compute the linear recurrence (4) using this partitioning of the hidden state. First, we note the dimensions of each weight and bias vector. Suppose the hidden dimension is h_{dim} and the embedding dimension if i_{dim}. Then we have the following:

1. $h^R, h^C \in \mathbb{R}^{h_{dim}} \implies h \in \mathbb{R}^{2*h_{dim}}$ and $W_h \in \mathbb{R}^{2h_{dim}*2h_{dim}}$.
2. $x_{ij} \in \mathbb{R}^{i_{dim}} \implies W_i \in \mathbb{R}^{2h_{dim}*i_{dim}}$.
3. $b \in \mathbb{R}^{2*h_{dim}}$.

Hence, we can partition the weight matrices and bias vector in (4) by defining the following partitions:

$$W_i = \begin{bmatrix} U^R \\ U^C \end{bmatrix} \tag{5}$$

$$W_h = \begin{bmatrix} V^{RR} & V^{RC} \\ V^{CR} & V^{CC} \end{bmatrix} \tag{6}$$

where each $U^* \in \mathbb{R}^{h_{dim}*i_{dim}}$ and each $V^{**} \in \mathbb{R}^{h_{dim}*h_{dim}}$.

Using the partitioning from (5) and (6) and partitioning the bias vector as $b = \begin{bmatrix} b^R \\ b^C \end{bmatrix}$, we see that the partitioned form of (4) is

$$\begin{bmatrix} Z^R \\ Z^C \end{bmatrix} = \begin{bmatrix} U^R x + V^{RR} h^R + V^{RC} h^C + b^R \\ U^C x + V^{CR} h^R + V^{CC} h^C + b^C \end{bmatrix}$$

Jointly updating the row and column states allows for the transfer of information between the row and column nodes via the weight matrices V^{RC} and V^{CR}, which we hypothesized would mitigate the issue of long-term memory – as the model is predicting entries row-wise, information early in the row-generation process becomes lossy without the joint update property of the LSTM with a partitioned hidden state.

To test this hypothesis, we trained the lobster graphs on two models: one which uses the single LSTM-update on the concatenated row and column states, and the other which uses two LSTMs that update the row and the column states independently, which corresponds to setting $V^{CR} = V^{RC} = 0$

in partitioning (6). As observed in Table 2, the LSTM joint update provided superior results on all observed metrics.

Table 2: Performance on updating the states simultaneously ("Joint") vs separately ("Independent")

Update Mode	Deg.	Clus.	Top Spec.	Wt. Spec.	MMDWt	Error
Joint	$2.46e^{-4}$	0.0	$9.98e^{-4}$	$8.51e^{-4}$	$4.93e^{-3}$	0.065
Independent	$3.27e^{-4}$	$6.20e^{-5}$	$2.98e^{-3}$	$2.46e^{-3}$	$6.40e^{-3}$	0.275

6.3 Log Likelihood and Training Objective Details

In this section, we provide details of the derivation of the training objective for the Adjacency LSTM. For readability, assume entries A_{ij} are conditioned on all prior entries; that is, $p(A_{ij}) \equiv p(A_{ij}|\{A_{kl}\}_{k<i;l<k})$

For each entry A_{ij}, we must decide if an edge connects nodes v_i and v_j by sampling $e_{ij} \sim$ Bernoulli(p_{ij}) and if so, sample a non-negative weight w_{ij} with prior $p(w_{ij})$. Hence, we note that the probabilities of a particular entry A_{ij} being 0 or weight w_{ij} are given by

$$p(A_{ij} = 0) = p(e_{ij} = 0) = 1 - p_{ij}$$
$$p(A_{ij} = w_{ij}) = p(e_{ij} = 1)p(w_{ij}|e_{ij} = 1) = p_{ij}p(w_{ij})$$

Hence, we may succinctly represent the probability of a particular entry of the adjacency matrix as

$$p(A_{ij} = w_{ij}) = (1 - p_{ij})^{1-e_{ij}} \big(p_{ij}p(w_{ij})\big)^{e_{ij}}$$

As the training objective is the log-likelihood over all entries A, we sum over the terms

$$\ell_{ij}\big(\theta; (e_{ij}, w_{ij})\big) = (1 - e_{ij}) \log(1 - p_{ij}) + e_{ij} \log p_{ij} + e_{ij} \log p(w_{ij})$$

where $e_{ij} \log p(w_{ij}) = 0$ if $e_{ij} = 0$. This yields the training objective for Adj-LSTM as

$$\mathcal{L}(\theta; E, W) = \log \prod_{i=1}^{n} \prod_{j=1}^{i-1} p(A_{ij}) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} \ell_{ij}(\theta; (e_{ij}, w_{ij}))$$

6.4 Information on Weights for Data

Weights were sampled from the following distributions:

- **Erdős–Rényi**: Weights were independently sampled from the standard normal distribution and transformed with the softplus function.
- **Tree**: Weights were sampled in a hierarchical manner. For each tee T_k, a mean μ_k was sampled uniformly between values 7 and 13. Given this mean, weights were sampled independently as $w_{ij} \sim \Gamma(\mu_k^2, \mu_k^{-1})$, where Γ is the gamma distribution.
- **3D Point Cloud**: Weights were computed as $w_{ij} = |n_u \cdot n_v|$, where n_u is the normal vector of the tangent plane consisting of all nodes adjacent to node u. (Neumann et. al, 2013)
- **Lobster**: Weights were independently sampled from the Beta distribution as $w_{ij} \sim$ Beta$(5, 15)$.

Tree Autoregressiveness. The hierarchical sampling scheme of the tree weights provided a means of testing for autoregressiveness in the models with respect to the edge weights. There were two main quantities of interest: the variance of weights pooled across all trees (Var(w_{ij})), and the variance of weights found in a single tree (Var$(w_{ij}|\mu_k)$)). Note a few preliminaries that are easily derived from their respective distributions

1. $\mu_k \sim$ Uniform$(7, 13) \implies \mathbb{E}(\mu_k) = 10$ and Var$(\mu_k) = 3$

2. $w_{ij} \sim \Gamma(\mu_k^2, \mu_k^{-1}) \implies \mathbb{E}(w_{ij}|\mu_k) = \mu_k$ and $\text{Var}(w_{ij}|\mu_k) = 1$

Importantly, the variance of weights found in each tree is free of the parameter μ_k.

Next, an application of iterative expectation and variance yield the mean and variance of weights pooled from all trees as

1. $\mathbb{E}(w_{ij}) = \mathbb{E}_\mu[\mathbb{E}_w(w_{ij}|\mu_k)] = \mathbb{E}_\mu(\mu_k) = 10.$

2. $\text{Var}(w_{ij}) = \mathbb{E}_\mu[\text{Var}_w(w_{ij}|\mu_k)] + \text{Var}_\mu[\mathbb{E}_w(w_{ij}|\mu_k)] = \mathbb{E}_\mu(\mathbb{1}) + \text{Var}_\mu(\mu_k) = 1 + 3 = 4$

Thus, to test for autoregressiveness in the models, we observe that weights pooled from all trees have variance $\text{Var}(w_{ij}) = 4$, whereas weights from a single tree have variance $\text{Var}(w_{ij}|\mu_k) = 1$.

6.5 Training Details

Hyperparameters. For Adj-LSTM, node states were parameterized with a hidden dimension of 128 and use a 2-layer LSTM. An embedding dimension of 32 was used to embed edge existence, and an embedding dimension of 16 was used to embed the weights. Positional encoding was used on the initialized row states.

Training Procedure. Both models were trained using the Adam Optimizer with weight decay and an initial learning rate of $1e^{-3}$.

For the tree and lobster data sets, BiGG-E was trained for 500 epochs and validated every 250 epochs. At epoch 100, the learning rate was decreased to $1e^{-5}$. The Adjacency LSTM was trained for 100 epochs on the tree data set and validated every 25 epochs, and trained for 300 epochs on the lobster data set and validated every 100 epochs. The learning rate was decayed to $1e^{-5}$ after 25 and 100 epochs, respectively.

For the Erdős–Rényi data set, BiGG-E was trained for 1000 epochs and validated every 250 epochs. At epoch 250, the learning rate was decreased to $1e^{-5}$. Due to timing, the Adjacency LSTM was only trained for 27 epochs. It is possible the model would improve generative quality of the graphs with more training time, which can be investigated in future work.

For the 3D Point Cloud data set, BiGG-E was trained for 3000 epochs and validated every 1000 epochs. At epoch 1000, the learning rate was decreased to $1e^{-5}$. Adj-LSTM was reported out of memory for this dataset.

Baseline Models. The Erdos-Renyi model baseline estimates were generated by first estimating the global probability of an edge existing between two nodes based on the training data, and then constructing Erdős–Rényi graphs with that probability of edge existence. Weights were sampled with replacement from all possible training weights in order to produce weighted graphs.

The BiGG+GCN model followed the same protocol outlined above with BiGG-E. For the convolutional network, two convolutions were used and each component was trained jointly on the same objective function used on BiGG-E.

6.6 Full Tables

Below are the tables for topological measures (Table 3) and weight measures (Table 4) from the experiments. The error reported for the tree and lobster datasets represent the proportion of graphs that were not bifurcating trees or lobster graphs. The "N-error" reported in the tree dataset represents the proportion of nodes across all graphs that have an incorrect number of edges, as we found most of the models produced trees where only one node did not meet the bifurcating property of the trees, which is not emphasized in the original error metric.

Table 3: Topological Measures

Datasets		Methods			
		Adj-LSTM	BiGG-E	BiGG+GCN	Erdos-Renyi
Erdős–Rényi	Deg.	0.437	$4.00e^{-3}$	$7.13e^{-3}$	$\mathbf{2.80e^{-3}}$
	Clus.	0.744	0.034	**0.014**	0.015
$\lvert V\rvert_{max} = 749$ (499)	Orbit	0.263	**0.061**	0.099	0.083
$\lvert E\rvert_{max} = 2846$ (1349)	Spec.	0.516	$2.81e^{-3}$	$4.26e^{-3}$	$\mathbf{2.06e^{-3}}$
Tree	Deg.	$\mathbf{6.63e^{-6}}$	$5.59e^{-5}$	$3.95e^{-5}$	0.269
	Spec.	$1.67e^{-3}$	$\mathbf{1.45e^{-3}}$	$8.55e^{-4}$	0.084
$\lvert V\rvert_{max} = 199$ (199)	Error	**0.125**	0.54	0.38	1.0
$\lvert E\rvert_{max} = 198$ (198)	N-Error	**0.003**	0.008	0.006	0.543
3D Point Cloud	Deg.	OOM	0.247	**0.070**	0.418
	Clus.	OOM	0.658	0.296	1.142
$\lvert V\rvert_{max} = 5037$ (1377)	Orbit	OOM	0.416	**0.293**	0.999
$\lvert E\rvert_{max} = 10886$ (3074)	Spec.	OOM	$1.64e^{-2}$	$\mathbf{8.25e^{-3}}$	0.089
Lobster	Deg.	$\mathbf{2.46e^{-4}}$	$8.28e^{-4}$	$4.84e^{-4}$	0.178
	Clus.	**0.0**	**0.0**	**0.0**	0.072
$\lvert V\rvert_{max} = 100$ (55)	Spec.	$\mathbf{9.98e^{-4}}$	$1.86e^{-3}$	$1.11e^{-3}$	0.192
$\lvert E\rvert_{max} = 99$ (54)	Orbit	$\mathbf{1.73e^{-3}}$	0.014	$5.87e^{-3}$	0.182
	Error	**.065**	.305	.115	1.0

Table 4: Weighted Measures

Datasets		Methods			
		Adj-LSTM	BiGG-E	BiGG+GCN	Erdos Renyi
Erdős–Rényi	Mean	-0.163	$3.16e^{-3}$	-0.022	$1.81e^{-3}$
	SD	0.983	1.007	0.998	1.002
$\mu_z = 0.0$	Spec.	0.469	$3.64e^{-3}$	$4.64e^{-3}$	$\mathbf{2.48e^{-3}}$
$\sigma_z = 1.0$	MMDWT	$4.40e^{-3}$	$1.80e^{-3}$	$\mathbf{1.40e^{-3}}$	$2.08e^{-3}$
Tree	$\bar{w}$	9.879	10.068	9.975	9.980
	s_w	1.955	1.962	1.955	1.949
$\mu_w = 10, \sigma_w = 2$	s_T	1.118	1.084	1.956	1.948
$\sigma_T = 1$	Spec.	$1.13e^{-3}$	$\mathbf{1.20e^{-3}}$	$2.09e^{-3}$	0.113
	MMDWT	$3.45e^{-3}$	$\mathbf{9.43e^{-4}}$	0.024	0.021
3D Point Cloud	Mean	OOM	0.226	0.234	0.224
	SD	OOM	0.415	0.164	0.068
$\bar{w} = 0.904$	Spec.	OOM	$1.73e^{-2}$	$\mathbf{9.50e^{-3}}$	0.112
$s_w = 0.213$	MMDWT	OOM	$\mathbf{2.34e^{-5}}$	$3.43e^{-3}$	$9.01e^{-3}$
Lobster	$\bar{w}$	0.249	0.250	0.252	0.254
	s_w	0.101	0.101	0.150	0.098
$\mu = 0.25$	Spec.	$8.51e^{-4}$	$\mathbf{7.86e^{-4}}$	$1.32e^{-3}$	0.270
$\sigma \approx 0.095$	MMDWT	$4.93e^{-3}$	$\mathbf{3.27e^{-3}}$	$5.09e^{-3}$	0.010