Towards Permutation-Invariant Graph Generation

Jenny Liu [1] **Aviral Kumar** [2] **Jimmy Ba** [1] **Kevin Swersky** [3]

Abstract

Learning to generate graph-structured data is a relatively new and challenging problem. Some of the most promising solutions so far rely on generating graphs via autoregressive processes; however, computing the likelihood under these models requires marginalizing over the set of all possible node orderings, a generally intractable problem. In this paper, we outline a permutation-invariant model based on variational autoencoders, with an expressive flow-based prior. We show results training the model in a 2-step process and demonstrate that this could be a promising approach for graph generation.

1. Introduction

Graph-structured data is ubiquitous in problems within science and engineering, and modeling graphs is an important component of prediction and reasoning within these domains. Machine learning has recently turned its attention to using graph-structured neural networks (Gori et al., 2005; Scarselli et al., 2009; Li et al., 2015; Kipf & Welling, 2016b; Gilmer et al., 2017) that can exploit the structure of relational systems to create more accurate and generalizable predictions. These can be used to predict molecular properties to aid in search and discovery (Duvenaud et al., 2015; Gilmer et al., 2017), or to learn physical properties of robots such that new structures can be controlled without re-learning a control policy (Wang et al., 2018).

While graph neural networks have enjoyed success in the supervised case, learned generative models of graphs are a relatively new and less explored area. Machine learning has been quite successful at generative modeling of complex domains such as images, audio, and text. However, relational data poses new and interesting challenges. For example, graphs are permutation-invariant: permutations of the nodes results in the same underlying structure.

[1]Vector Institute, University of Toronto, Canada [2]UC Berkeley, California, USA [3]Google Research, Toronto, Canada. Correspondence to: Jenny Liu <jyliu@cs.toronto.edu>.

Presented at the ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data

One of the most successful approaches so far is to model the graph using an auto-regressive process (Li et al., 2018; You et al., 2018). These generate each node in sequence, and for each newly generated node, the corresponding edges to previously generated nodes are also created. In theory, this is capable of modeling the full joint distribution, but computing the full likelihood requires marginalizing over all possible node-orderings. Sequential generation using RNNs also potentially suffers from the issue of long-range dependencies for large graphs.

In this paper, we propose an approach to modeling graphs in a permutation-invariant way. We use a reversible formulation of graph neural networks (Kumar et al., 2018) to form a distribution over sets of related points. This is a direct extension of normalizing flows (Rezende & Mohamed, 2015; Dinh et al., 2017) to continuous spaces involving relational data. As our overall objective is to produce a generative model over graph structures, which are inherently discrete, we augment our model with a graph auto-encoder. The auto-encoder generates an embedding space that the normalizing flow models, and the decoder generates adjacency matrices from the embeddings. We train each component separately, and combine them at inference time. The result is a permuation-invariant generative model that is well suited to parallel computing architectures.

2. Methods

2.1. Graph Neural Networks

Graph Neural Networks (GNNs) or Message Passing Neural Nets (MPNNs) (Gilmer et al., 2017) are a generalization/unification of a number of neural net architectures on graphs used in literature for a variety of tasks ranging from molecular modeling to network relational modeling. In general, MPNNs have two phases in the forward pass – a message passing (MP) phase and a readout phase (R). The MP phase runs for T time steps, and is defined in terms of message generation functions M_t and vertex update functions U_t. During each step in the message passing phase, hidden node features $\mathbf{h}_t^{(v)}$ at each node in the graph are updated based on messages $\mathbf{m}_{t+1}(v)$ according to $\mathbf{m}_{t+1}(v) = \sum_{u \in \mathcal{N}(v)} M_t(\mathbf{h}_t^{(v)}, \mathbf{h}_t^{(u)}, \Omega_{u,v})$ and $\mathbf{h}_{t+1}^{(v)} = U_t(\mathbf{h}_t^{(v)}, \mathbf{m}_{t+1}(v))$, where $\mathcal{N}(v)$ denotes the set

of neighbours to node v in the graph. The readout phase usually converts the final node embeddings generated at the end of message passing into task specific features. For example, to have one embedding for the entire graph, we have $y_{\mathcal{G}} = \mathsf{Agg}(\{\mathbf{h}_T^{(v)}\})$, where $\mathsf{Agg}(\cdot)$ is an aggregation function.

One particularly useful aggregation function is graph attention (Velikovi et al., 2018), which uses attention (Bahdanau et al., 2015; Vaswani et al., 2017) to weight the messages from adjacent nodes. This involves computing an attention coefficient α between adjacent nodes using a linear transformation W, an attention mechanism a, and a nonlinearity σ,

$$e_{t+1}^{(v,u)} = a(W\mathbf{h}_t^{(v)}, W\mathbf{h}_t^{(u)})$$
$$\alpha_{t+1}^{v,u} = \frac{\exp(e_{t+1}^{(v,u)})}{\sum_{w\in\mathcal{N}(v)}\exp(e_{t+1}^{(u,w)})}$$
$$\mathbf{m}_{t+1}^{(v)} = \sigma\Big(\sum_{u\in\mathcal{N}(v)} \alpha_{t+1}^{(v,u)} M(\mathbf{h}_t^{(v)}, \mathbf{h}_t^{(u)}, \Omega_{u,v})\Big)$$

Multi-headed attention (Vaswani et al., 2017) applies attention with multiple weights W and concatenates the results.

2.2. Reversible Graph Neural Networks (GRevNets)

GRevNets (Kumar et al., 2018) are a family of reversible message passing neural network models. In order to convert a GNN/MPNN to a Reversible MPNN (GRevNet), the node feature matrix is split into two parts along the feature dimension–call them $H_t^{(0)}$ and $H_t^{(1)}$ respectively. Now for a particular node in the graph v, the two parts of its features at time t in the message passing (MP) phase are called $\mathbf{h}_t^0$ and $\mathbf{h}_t^1$ respectively. [$\mathsf{concat}(\mathbf{h}_t^0, \mathbf{h}_t^1) = \mathbf{h}_t^{(v)}$]

One step of the message passing procedure is broken down into into two intermediate steps, each of which is denoted as a half-step. $F(.)$ and $G(.)$ denote two instances of the 1-step MP transformation (both F and G consist of applying M_t once and then U_t once – that is, generate messages from each nodes, send it to neighbours and update node hidden states based on the aggregated message received) on the node features given the graph adjacency matrix Ω. Figure 1 depicts the procedure in detail.

$$\mathbf{h}_{t+0.5}^0 = \mathbf{h}_t^0 + F(\mathbf{h}_t^1) \qquad \mathbf{h}_{t+1}^0 = \mathbf{h}_{t+0.5}^0$$
$$\mathbf{h}_{t+0.5}^1 = \mathbf{h}_t^1 \qquad \mathbf{h}_{t+1}^1 = \mathbf{h}_{t+0.5}^1 + G(\mathbf{h}_{t+0.5}^0)$$

Reversibility: It is easy to see that this architecture is reversible. For any time step during message passing, if we know the values of $\mathbf{h}_t^0$ and $\mathbf{h}_t^1$, then we can simply recover the previous time step node features by reversing the computation: $\mathbf{h}_{t-1}^0 = \mathbf{h}_t^0 - F(\mathbf{h}_t^1 - G(\mathbf{h}_t^0))$ and $\mathbf{h}_{t-1}^1 = \mathbf{h}_t^1 - G(\mathbf{h}_t^0)$. In the same spirit as flows[?], we can use the chain rule of statistics to give us the rule for exact density transformation. So, if we assume $\mathbf{h}_t \sim P(\mathbf{h}_t)$, then the density is given by: $p(\mathbf{h}_t) = \det\left|\frac{d\mathbf{h}_{t-1}}{d\mathbf{h}_t}\right| p(\mathbf{h}_{t-1})$, where the Jacobian is given by a lower triangular matrix with determinant 1, hence making density computations tractable. In practice, we also apply a scaling factor, thereby making the model non-volume preserving by modifying the equations as: $\mathbf{h}_{t+0.5}^0 = \exp(F_2(\mathbf{h}_t^1)) \cdot \mathbf{h}_t^0 + F_1(\mathbf{h}_t^1)$ and $\mathbf{h}_{t+1}^0 = \mathbf{h}_{t+0.5}^0$ and analogously for the other half. This scales the Jacobian by the product of the scaling factor in each half. For more details please refer to (Dinh et al., 2017).

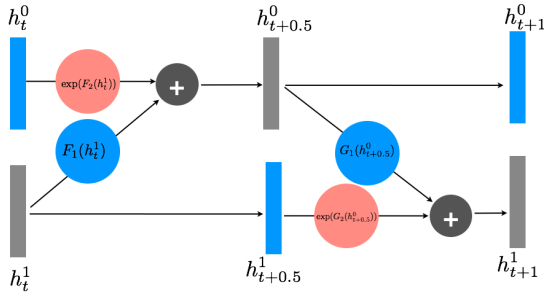


Figure 1. Architecture of 1 step Message Passing in a GRevNet: $\mathbf{h}_t^0$, $\mathbf{h}_t^1$ denote the two parts of the node-features of a particular node, $F_1(.), F_2(.)$ and $G_1(.), G_2(.)$ are 1-step MP transforms consisting of applying M_t and U_t once each. The scaling functions (F_2, G_2) are shown in red, whereas the translation functions F_1, G_1 are shown in blue.

Owing to its reversibility, a GRevNet (Kumar et al., 2018) that can preserve bijection between the input features and output features can be used to model expressive priors for learning distributions over the graph structure via auto-encoding optimization schemes. As the computational units in a GRevNet are GNN message passing modules, such a reversible generative model is also permutation-invariant. In particular, if the GRevNet maps from one continuous space to another using the relational structure induced by an adjacency matrix, then the GRevNet can be used as a graph-based normalizing flow. We use this insight to design a generative model on graphs which we describe next.

2.3. Proposed Generative Model

The aim of learning graph generative models is to maximize the marginal likelihood of graphs $P(\mathcal{G})$ under the training data. Our objective is to train a generative model of graph structures, an inherently discrete problem. Our strategy to solve this is to use a two-step process: (1) train a permutation-invariant graph auto-encoder to create a graph encoder that embeds graphs into a continuous space; (2) train a GRevNet to model the distribution of the graph embeddings, and use the decoder to generate graphs. Each

stage is trained separately.

In the absence of a known graph structure for generation, we use a fully connected graph neural network. This allows the model to learn how to organize nodes in order to match a specific distribution. However, this poses a problem for certain aggregation functions like sum and mean, where the messages from each node will have to contend with the messages from every other node. If there is a salient piece of information being sent from one node to another, then it could get drowned out by less informative messages. Instead, we opt to use graph attention, which allows each node to choose the messages that it deems to be the most informative.

We now describe how we perform structure learning. In contrast to GraphVAE (Kipf & Welling, 2016a), which generates a single vector to model the entire graph, we instead embed a set of nodes in a graph jointly, but each node is mapped to its own embedding vector. This avoids the issue of having to run matching in the decoder.

We propose a graph-autoencoder that takes in a graph $\mathcal{G}$ and reconstructs the elements of the adjacency matrix, A, where $A_{ij} = 1$ if node v_i has an edge connecting it to node v_j, and 0 otherwise. We focus on undirected graphs, meaning that we only need to predict the upper (or lower) triangular portion of A, but this methodology could easily extend to directed graphs. The encoder takes in a set of node features $H \in \mathbb{R}^{N \times d}$ and an adjacency matrix $A \in \{0,1\}^{N \times \frac{N}{2}}$ ($\frac{N}{2}$ since the graph is undirected) and outputs a set of node embeddings $X \in \mathbb{R}^{N \times k}$. The decoder takes these embeddings and outputs a set of edge probabilities $\hat{A} \in [0,1]^{N \times \frac{N}{2}}$. For parameters $\boldsymbol{\theta}$, we use the binary cross entropy loss function, $\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \sum_{j=1}^{\frac{N}{2}} A_{ij} \log(\hat{A}_{ij}) + (1 - A_{ij}) \log(1 - \hat{A}_{ij})$ We use a relatively simple decoder. Given node embeddings $\mathbf{x}_i$ and $\mathbf{x}_j$, our decoder outputs the edge probability as $\hat{A}_{ij} = \frac{1}{1+\exp(C(\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 - 1))}$, where C is a temperature hyperparameter, set to 10 in our experiments. This reflects the idea that nodes that are close in the embedding space should have a high probability of being connected.

The encoder is a standard GNN with multi-head dot-product attention, that uses the adjacency matrix A as the edge structure (and no additional edge features). In order to break symmetry, we need some way to distinguish the nodes from each other. If we are just interested in learning structure, then we do not have access to node features, only the adjacency matrix. In this case, we generate node features H using random Gaussian variables $\mathbf{h}_i \sim \mathcal{N}(0, \sigma^2 I)$, where we use $\sigma^2 = 0.3$. This allows the graph network to learn how to appropriately separate and cluster nodes according to A. We generate a new set of random features each time we encode a graph. This way, the graph can only rely on the features to break symmetry, and must rely on the graph structure to generate a useful encoding.

Putting the GRevNet together with the graph encoder, we map training graphs from H to X and use this as training inputs for the model. Generating involves sampling $Z \sim \mathcal{N}(0, I)$ followed by inverting the model, $X = f^{-1}(Z)$, and finally decoding X into A and thresholding to get binary edges.

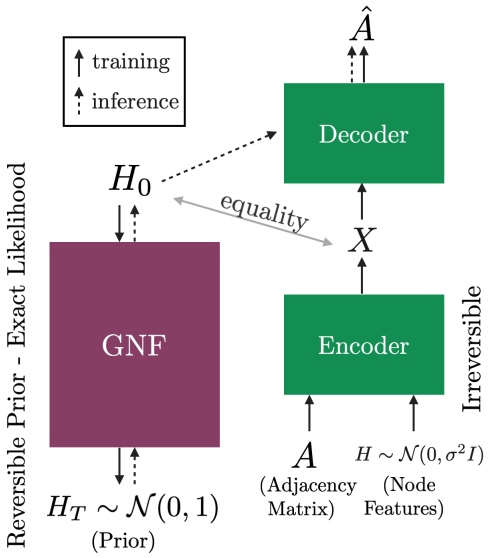


Figure 2. Pipeline of our proposed graph generation framework. We train a graph auto-encoder to learn node embeddings that can reconstruct the given adjacency matrix. We then train a GRevNet to model this distribution over node embeddings.

3. Experiments

3.1. Flow Priors

To motivate using the GRevNet as a more expressive prior, we trained a VAE on binarized MNIST and compared the performance of a RealNVP versus Gaussian prior. We found that the RealNVP prior achieves a better log likelihood.

Table 3. Binarized MNIST evaluation of a VAE trained with three different priors. MIXTURE OF GAUSSIANS has 100 mixture components.

PRIOR	NLL
GAUSSIAN	92.9
MIXTURE OF GAUSSIANS	90.7
REALNVP	89.8

3.2. GRevNet Flows

We next investigate the expressiveness of our GRevNet flow prior by training a GRevNet model on 3 synthetic datasets for structured density estimation.

Datasets: In MIXTURE OF GAUSSIANS (MOG), each training example is a set of 4 points in a square configuration.

Table 1. Comparison of train negative log likelihood (NLL) for RealNVP and GRevNet on 3 synthetic datasets.

	DATASET		
ARCHITECTURE	MOG (NLL)	MOG RING (NLL)	6-HALF MOONS (NLL)
REALNVP	4.2	5.2	-1.2
GREVNET	3.6	4.2	-1.7

Table 2. Train and test binary cross-entropy (CE), averaged over the total number of nodes. TOTAL # INCORRECT EDGES measures the number of incorrect edge predictions (either missing or extraneous) in the reconstructed graphs over the entire dataset. TOTAL # EDGES lists the total number of edges in each dataset.

	BINARY CE		TOTAL # INCORRECT EDGES		TOTAL # EDGES	
DATASET	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST
EGO-SMALL	9.8E-4	11E-04	24	32	3758	984
COMMUNITY-SMALL	5E-4	7E-04	10	2	1329	353

Each point is drawn from a separate isotropic Gaussian, so no two points should land in the same area. MIXTURE OF GAUSSIANS RING (MOG RING) takes each example from MOG and rotates it randomly about the origin, creating an aggregate training distribution that forms a ring. 6-HALF MOONS interpolates the original half moons dataset using 6 points with added noise. Visualizations of the training data are provided in the Appendix.

We train a GRevNet where each of F and G is a self-attention module followed by an MLP, and we train an analogous RealNVP model with the same number of steps and the same MLP architecture. The only difference is the RealNVP model does not have self-attention, as nodes are updated independently from one another.

As shown in Table 1, GREVNET is able to achieve significantly lower negative log likelihood than REALNVP. As GREVNET allows message passing between nodes, it can more accurately model the dependencies between nodes, whereas by design, REALNVP cannot do this. We show a visualization of the generated samples in Figures 4-6 in the Appendix.

3.3. Graph Generation

Datasets: We run our graph generation model on two datasets, COMMUNITY-SMALL and EGO-SMALL from GraphRNN (You et al., 2018). COMMUNITY-SMALL is a procedurally-generated set of 100 2-community graphs, where $12 \leq |V| \leq 20$. EGO-SMALL is a set of 200 graphs, where $4 \leq |V| \leq 18$, drawn from the larger Citeseer network dataset (Sen et al., 2008).

3.3.1. GRAPH AUTO-ENCODER

We train a graph auto-encoder with attention. Every training epoch, we generate new Gaussian noise features for each graph as input to the encoder.

Table 2 shows that our auto-encoder generalizes well to unseen test graphs, with a small gap between train and test cross-entropy. The total # of incorrect edges metric shows that the model achieves good test reconstruction on EGO-SMALL and near-perfect test reconstruction on COMMUNITY-SMALL.

3.3.2. GREVNET PRIOR

Our trained auto-encoder gives us a distribution over node embeddings that are useful for graph reconstruction. We then train a GRevNet to maximize the likelihood of these embeddings using an isotropic Gaussian as the prior. Once trained, at generation time we can flow N random Gaussian embeddings sampled from the prior to N node embeddings by running the GRevNet forward. These node embeddings will then describe a graph adjacency when run through the decoder.

In Figure 3, we visualize the training distribution and generated samples of our model on the two datasets. In the Appendix we give a qualitative comparison between our model and the state-of-the-art GraphRNN baseline.

4. Conclusion

We introduce a generative extension of reversible graph neural networks (GRevNets) based on normalizing flows. Our model is capable of learning distributions over related points in a continuous space. We combine this with a graph auto-encoder to generate adjacency matrices, and train both components independently. Our model is permutation-invariant, while still competitive with auto-regressive graph generation. In future work, we plan to focus on training the entire system in an end-to-end fashion under a variational auto-encoding framework (Kingma & Welling, 2014).

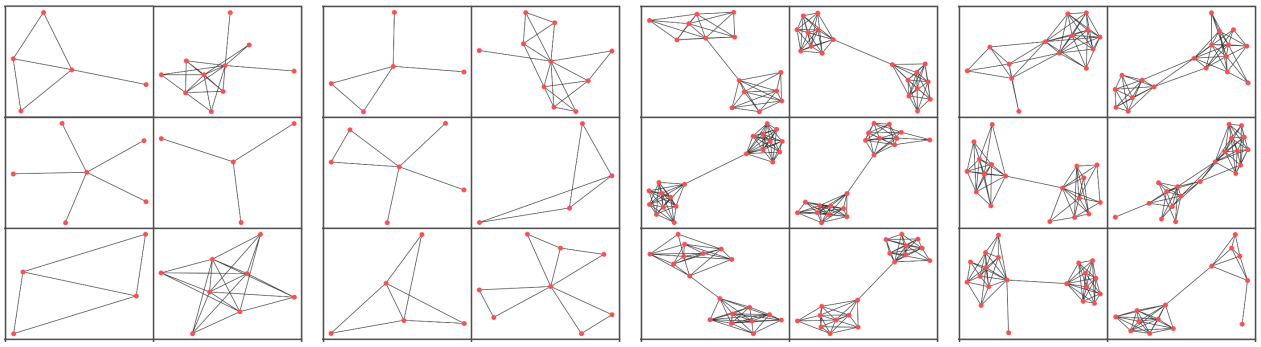


Figure 3. From Left to Right: EGO-SMALL training data, generated samples from GRevNet, COMMUNITY-SMALL training data, generated samples from GRevNet. All examples picked randomly.

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*, 2015.

De Cao, N. and Kipf, T. MolGAN: An implicit generative model for small molecular graphs. *arXiv e-prints*, art. arXiv:1805.11973, May 2018.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *International Conference on Learning Representations*, 2017.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/gilmer17a.html.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *J. Mach. Learn. Res.*, 13:723–773, March 2012. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2188385.2188410.

Kearnes, S., Li, L., and Riley, P. Decoding Molecular Graph Embeddings with Reinforcement Learning. *arXiv e-prints*, art. arXiv:1904.08915, Apr 2019.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *CoRR*, abs/1611.07308, 2016a.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016b.

Kumar, A., Ba, J., Kiros, J., and Swersky, K. Grevnet: Improving graph neural nets with reversible computation. In *NeuRIPS Relational Representation Learning Workshop, NeurIPS 2018*, Montreal, Canada, 2018. URL https://drive.google.com/file/d/1UYsTSnyKjl6MAox9vwGtV77wB_3vMavR/view.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning Deep Generative Models of Graphs. *arXiv e-prints*, art. arXiv:1803.03324, Mar 2018.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International Conference on Machine Learning*, volume 37, pp. 1530–1538, 2015.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., and Eliassi-Rad, T. Collective classification in network data. Technical report, 2008.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. 2017. URL https://arxiv.org/pdf/1706.03762.pdf.

Velikovi, P., Cucurull, G., Casanova, A., Romero, A., Li, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Wang, T., Liao, R., Ba, J., and Fidler, S. Nervenet: Learning structured policy with graph neural networks. *ICLR*, 2018.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. GraphRNN: Generating realistic graphs with deep auto-regressive models. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5708–5717, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/you18a.html.

5. Appendix

5.1. Related Work

Most existing approaches to graph generation draw inspiration from the success of deep generative models in computer vision and natural language processing. Autoregressive Models such as GraphRNN (You et al., 2018) and (Li et al., 2018) use a BFS-ordering over the nodes of a graph and SMILES representation for molecules respectively to define an injection between graphs to the space of sequences of nodes of the graph, and train an autoregressive model to sample edges and nodes for the graph. However, marginalizing over all possible BFS orderings of a graph during training is intractable and randomization over permutations during training is still not enough to generalize to all of them is a challenge in this model. In this work, we used GRevNet flows to provide permutation-invariant modelling of graphs.

(Kipf & Welling, 2016a) cast the graph structure learning problem in a probabilistic auto-encoding framework, training the model with reconstruction loss treating graph structure generation as an optimization problem in continuous space analogous to image generation using VAEs. This model over-generates nodes and has to perform a matching in the decoder as a result. Concurrent to us, (Kearnes et al., 2019) used techniques inspired from approximate dynamic programming to train a sequential decoder for graphs in a similar way to an autoregressive model trained without supervised learning.

Further, other works inspired from Generative Adversarial Networks (GANs) train discriminators on sets of real and fake graphs (De Cao & Kipf, 2018). In our work, we consider a graph structure learning problem similar to (Kipf & Welling, 2016a) but with more expressive prior features (h) which are outputs of a fully reversible and explicit density generative model and are much more expressive than just a factored gaussian posterior.

5.2. GRevNet Samples

In Figure 4 we provide a visualization of the training data and generated samples for the structured density estimation experiments as described in Section 3.2.

5.3. Graph Generation Quantitative Evaluation

We evaluate our model by using the quantitative evaluation technique in GraphRNN (You et al., 2018), which calculates the MMD distance (Gretton et al., 2012) between the generated graphs and a previously unseen test set on three statistics based on degrees, clustering coefficients, and orbit counts. We use the implementation of GraphRNN provided by the authors to train their model and their provided evaluation script to generate all quantitative results, which are provided in Table 4.

5.4. Experimental Setup

5.4.1. Flow Priors

We used an encoder with 5 convolutional layers and a decoder with 5 deconvolutions. The RealNVP prior consisted of 4 coupling layers, where each of F and G is an MLP with 2 layers of 512 hidden units each. We trained for 25k steps and used the Adam Optimizer with a learning rate of 1e-03.

5.4.2. GRevNet Flows

We used 12 GRevNet coupling layers, and for each MLP we had 5 fully-connected layers of 256 hidden units each with ReLu nonlinearities. We trained for 15k steps using Adam with a learning rate of 1e-04.

5.4.3. Graph Auto-encoder

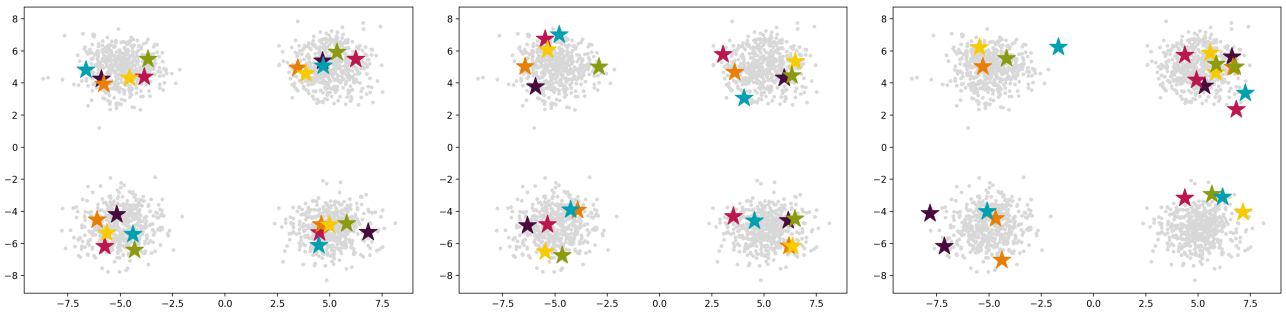
We used a GNN with 10 timesteps, and each MLP had 3 layers of 2048 hidden units with ReLu nonlinearities. We used Adam with a learning rate of 5e-04.

5.4.4. GRevNet Graph Prior

We used a GRevNet with 12 timesteps, and each MLP had 3 layers of 2048 hidden units with ReLu nonlinearities. We used Adam with a learning rate of 1e-05.

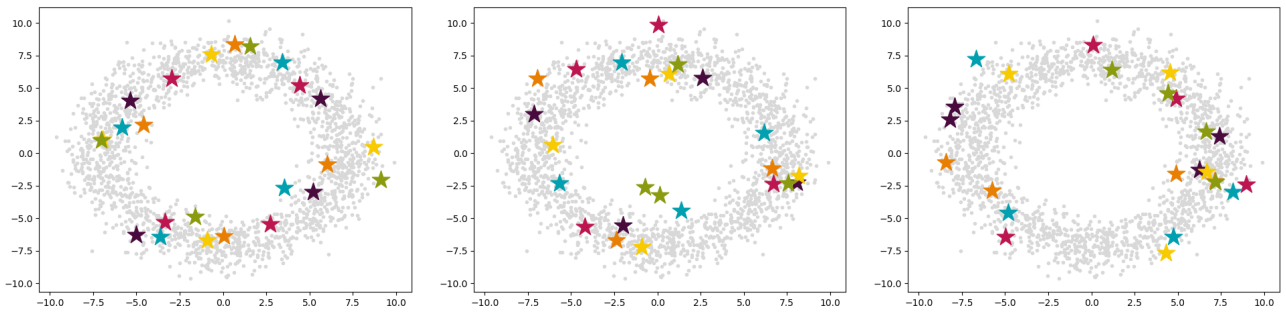
	COMMUNITY-SMALL			EGO-SMALL		
MODEL	DEGREE	CLUSTER	ORBIT	DEGREE	CLUSTER	ORBIT
GRAPHRNN	**0.03**	**0.01**	**0.01**	0.04	0.05	0.06
GREVNET	0.12	0.15	0.02	**0.01**	**0.03**	**0.0008**

Table 4. Graph generation results depicting MMD for various graph statistics between the test set and generated graphs. The results are obtained by comparing the test set with 1024 generated graphs. We trained and evaluated the result over 5 separate runs per model.



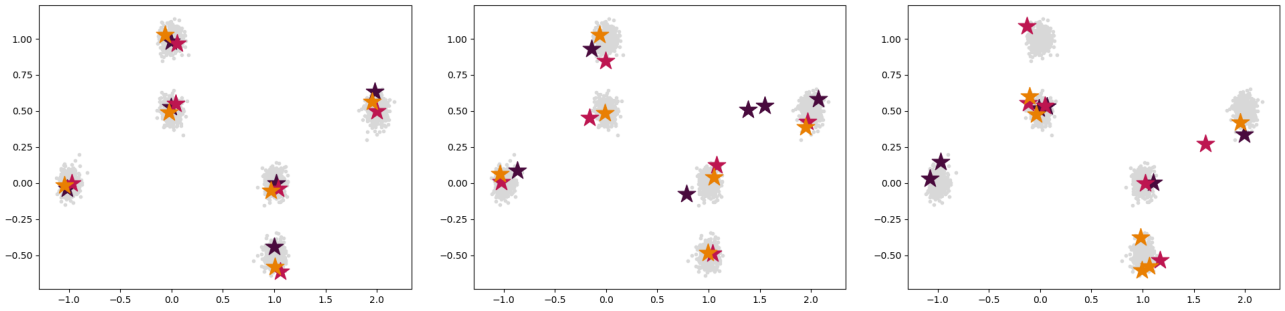
(a) Training examples (b) GRevNet samples (c) RealNVP samples

Figure 4. (a) shows the aggregate training distribution for the MIXTURE OF GAUSSIANS (MOG) dataset in gray, as well as 5 individual training examples. Each training example is shown in a different color and is a structured set of nodes where each node is drawn from a different Gaussian. (b) and (c) each show 5 generated samples from GRevNet and RealNVP, selected randomly, where each sample is a different color. The GRevNet learns to generate structured samples where each node resembles a sample from a different Gaussian, while RealNVP cannot.



(a) Training examples (b) GRevNet samples (c) RealNVP samples

Figure 5. (a) shows the aggregate training distribution for the MIXTURE OF GAUSSIANS RING (MOG RING) dataset in gray, as well as 5 individual training examples. Each training example is shown in a different color and is a structured set of nodes in a square configuration. (b) and (c) each show 5 generated samples from GRevNet and RealNVP, selected randomly, where each sample is a different color. GRevNet makes one mistake with the green sample, but otherwise all the other samples have square configurations, whereas RealNVP generates several examples (purple, green, orange) that are not square-like.



(a) Training examples (b) GRevNet samples (c) RealNVP samples

Figure 6. (a) shows the aggregate training distribution for the 6-HALF MOONS dataset in gray, as well as 3 individual training examples. Each training example is shown in a different color and is a structured set of nodes where each node is drawn from a different cluster. (b) and (c) each show 3 generated samples from GRevNet and RealNVP, selected randomly, where each sample is a different color. GRevNet can sometimes generate perfect samples, as in the orange and pink examples.