Graph Generation with Energy-Based Models

Jenny Liu [1 2] **Will Grathwohl** [1 2] **Jimmy Ba** [1 2] **Kevin Swersky** [3]

Abstract

We present a set of novel, energy-based models built on top of graph neural networks (GNN-EBMs) to estimate the unnormalized density of a distribution of graphs. GNN-EBMs can generate graphs implicitly via MCMC sampling. We compare the performance of GNN-EBMs trained using 3 different estimators: pseudolikelihood, conditional noise contrastive estimation, and persistent contrastive divergence (PCD). We find that all 3 estimators result in models that generalize well, while models trained with PCD generate samples that are competitive with state-of-the-art baselines. Finally, we discuss the potential of GNN-EBMs beyond generation for diverse tasks such as semi-supervised learning and outlier detection.

1. Introduction

Recent work on learning generative models has focused on distributions over graphs. Many datasets can be succinctly expressed in graph form, and generative models over these structures can be used in problems such as molecule design (Gilmer et al., 2017), robotics (Wang et al., 2018), recommender systems (Ying et al., 2018), and more. There have been a number of proposals based on adapting generative models from other domains such as vision and natural language. These include auto-regressive models (You et al., 2018b), normalizing flows (Liu et al., 2019), and variational auto-encoders (Simonovsky & Komodakis, 2018).

In this paper, we propose *energy-based models* for modelling distributions over graph structures. EBMs have a rich history in generative modelling (Hinton et al., 2006b;a) and have recently shown promising results and desirable properties for image generation (Du & Mordatch, 2019; Grathwohl et al., 2019). EBMs are flexible: they only require a mapping from a graph structure to a scalar. Many domains have important constraints; for example, in molecule design the generated molecules must be physically possible. EBMs allow a natural specification of constraints through the energy function. Closely related to our model are recent proposals based on denoising auto-encoders (Seff et al., 2019), and score function estimation (Niu et al., 2020); both implicitly define an EBM, whereas we define ours explicitly.

However, EBMs also require the estimation of an often intractable normalization constant. This poses challenges for parameter estimation, likelihood evaluation, and sampling. We explore proposed estimators from the literature, apply them to graph data, and show they generate robust energy landscapes. We further describe our sampling procedure, and approximation strategies to improve computational efficiency. We find that GNN-EBMs trained with persistent contrastive divergence (Tieleman, 2008) in particular generate samples competitive with current state-of-the-art baselines. Finally, we discuss future research directions towards improving and scaling the EBM approach.

2. Background

2.1. Graph Neural Networks

Graph Neural Networks (GNNs) or Message Passing Neural Nets (Scarselli et al., 2008; Gilmer et al., 2017) typically use message passing (MP) steps to update node features, followed by global pooling to construct a single graph embedding. One useful MP aggregation is graph attention (Veličković et al., 2017), which uses attention (Vaswani et al., 2017; Bahdanau et al., 2015) to weight messages.

2.2. Energy-Based Models

Energy-based models (EBMs) define a distribution over the input data as $p_\theta(\mathbf{x}) = \frac{e^{f_\theta(\mathbf{x})}}{Z(\theta)}$. Given an input domain Ω, EBMs assign each point an unnormalized log probability $f_\theta(\mathbf{x}) : \Omega \to \mathbb{R}$. Here, θ represents the model parameters, and $Z(\theta) = \sum_{\mathbf{x}\in\Omega} \exp(f_\theta(\mathbf{x}))$ is the normalization constant. EBMs get their name from the so-called *energy function*, which is simply $-f$. This function fully specifies the distribution. Because $Z(\theta)$ is typically intractable, most EBMs cannot be trained by maximum likelihood and instead must rely on alternative estimators. We will describe several such estimators in the remainder of this section.

Pseudolikelihood (Besag, 1975) approximates $p(\mathbf{x}; \theta)$ as:

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

ICML Workshop on Graph Representation Learning and Beyond (GRL+), 2020.

$$p(\mathbf{x};\theta) = \prod_i p(x_i|x_1,...,x_{i-1}) \approx \prod_i p(x_i|x_{\neg i};\theta)$$

where $\mathbf{x} \in \mathbb{R}^n$ and $x_{\neg i}$ denotes $\mathbf{x}$ with its ith element removed. With pseudolikelihood, we only need to be able to compute the conditional distribution of each component, fixing all others. The log-pseudolikelihood objective is

$$\ell_{\text{PL}}(\theta;\mathbf{x}) = \sum_{i=1}^{n} \log \frac{e^{f(\mathbf{x};\theta)}}{e^{f(\mathbf{x}_{\neg i};\theta)} + e^{f(\mathbf{x};\theta)}}$$

Conditional noise contrastive estimation (CNCE) (Ceylan & Gutmann, 2018) and noise contrastive estimation (NCE) (Gutmann & Hyvärinen, 2010) reformulate the estimation problem as one of discriminating between true data and noise samples. NCE is more efficient when the noise distribution closely resembles the data distribution, and CNCE achieves this by using corrupted data as the noise distribution.

Using $\mathbf{x}'$ to denote corrupted data, the CNCE loss of an observation is as follows:

$$\begin{aligned}\ell_{\text{CNCE}}(\theta;\mathbf{x}) &= \log[1+\exp(-F(\mathbf{x},\mathbf{x}';\theta))] \\ F(\mathbf{x},\mathbf{x}';\theta) &= f(\mathbf{x};\theta) - f(\mathbf{x}';\theta) \\ &\quad + \log p_c(\mathbf{x}|\mathbf{x}') - \log p_c(\mathbf{x}'|\mathbf{x}) \quad (1)\end{aligned}$$

where $p_c(\mathbf{x}'|\mathbf{x})$ is the probability of observing corrupted sample $\mathbf{x}'$ given data $\mathbf{x}$.

Persistent contrastive divergence (PCD) (Younes, 1989; Tieleman, 2008) trains by approximating the gradient of the log-likelihood with:

$$\nabla_\theta \log p(\mathbf{x};\theta) = \nabla_\theta f(\mathbf{x};\theta) - \mathbb{E}_{\tilde{\mathbf{x}}\sim p(\mathbf{x};\theta)}\left[\nabla_\theta f(\tilde{\mathbf{x}};\theta)\right]$$

where $\tilde{\mathbf{x}}$ are samples drawn from the model distribution. Computing the expectation in this estimator is intractable in general, however we can approximate it via Monte Carlo sampling. Contrastive divergence (Hinton, 2002; Carreira-Perpinan & Hinton, 2005) runs a short Markov chain Monte Carlo (MCMC) chain initialized from $\mathbf{x}$. PCD instead maintains a persistent chain by initializing each short MCMC chain from the previous one. This provides better exploratory behavior, and lower bias.

3. Methods

3.1. GNN-EBMs

Here we present GNN-EBMs, which use GNNs to define graph distributions under the energy-based framework. We describe the methods we use to adapt each estimator to graph-structured data. We note that these methods can be easily applied to a wide range of GNN architectures, though we are able to use standard components like Veličković et al. (2017) as described in Section 4.1 to attain useful models. We include the algorithms described in this section in the supplementary material, Section 7.1.

3.2. Notation

Our training set consists of N undirected graphs, $(\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_N)$. A graph $\mathcal{G}$ consists of a set of nodes $\mathcal{V}_\mathcal{G}$ and a set of edges $\mathcal{E}_\mathcal{G}$. Its structure can also be expressed as an upper triangular adjacency matrix $A \in \{0,1\}^{|\mathcal{V}_\mathcal{G}|\times|\mathcal{V}_\mathcal{G}|}$. We use $A_{\overline{ij}}$ to denote an adjacency matrix A where the entry in the ith row and jth column has been flipped.

We learn function $f(\mathcal{G};\theta)$, where $f : \mathcal{G} \mapsto \mathbb{R}$. $f(\mathcal{G};\theta)$ is defined by a GNN that maps graphs to unnormalized log probabilities, or negative energies. To slightly abuse notation, we also allow f to take in an adjacency matrix, so $f(A;\theta)$ is valid.

3.3. Pseudolikelihood

For graphs, we can model the elements of a graph's adjacency matrix as a collection of binary-valued random variables. Computing the conditional for each element is easy, as we only need the energy of the original adjacency and the energy of the adjacency with that element flipped. We can express this conditional as:

$$p(A_{ij}) = \log \frac{e^{f(A;\theta)}}{e^{f(A_{\overline{ij}};\theta)} + e^{f(A;\theta)}}$$

Thus, we can interpret pseudolikelihood as maximizing the conditional probability of each element in a graph's adjacency matrix, with all other elements fixed.

3.4. Conditional Noise Contrastive Estimation

To adapt CNCE to graph data, we must decide on the appropriate conditional noise distribution $p_c(\mathcal{G}'|\mathcal{G})$, where we use $\mathcal{G}'$ to denote a noise graph. In most cases, this distribution is not permutation invariant, and doing the proper marginalization over all permutations is intractable for larger graphs. However, if we define a symmetric conditional noise distribution, then the two conditional noise terms cancel each other out in Equation 1.

In Algorithm 1, we define a simple, symmetric way to generate noise graphs. By independently sampling noise parameter p from a Beta distribution for every new noise graph, we induce more diversity in the samples. We can also tune Beta to generate noise samples that are closer to the data.

3.5. Persistent Contrastive Divergence

To adapt PCD to graph data, we need to generate $\tilde{\mathcal{G}}$ from the model distribution. We can do this using Gibbs sampling as described in Algorithm 2. While previous approaches for training restricted Boltzmann machines have relied on block Gibbs sampling to parallelize this process (Fischer & Igel, 2014), we must do sequential sampling. This potentially slows down training as a full Gibbs step involves iterating through all $\binom{|\mathcal{V}|}{2}$ entries of the adjacency. To mitigate this, we carefully tune the number of Gibbs "mini-steps" we

use over the adjacency entries to balance training time with approximation accuracy.

3.6. Implicit Generation

Exact sampling from the GNN-EBM is intractable, as it is in most EBMs. Instead, we rely on sampling through MCMC. Here, we describe our sampling strategies. Note that our sampling methods are currently designed for a fixed number of nodes, N. We sample this from the empirical training distribution, and then run MCMC over the edges to sample adjacency matrices.

Gibbs Sampling. As described in Algorithm 2, starting with some initial graph, we can use Gibbs sampling for generation by continuously iterating through its adjacency matrix and sampling from the full conditional distribution for each edge, which in this case corresponds to a Bernoulli distribution. Given enough iterations, this process will eventually converge to sampling from the true model distribution.

Greedy Generation. Although Gibbs sampling will eventually generate samples from the true distribution, it may take an impractical number of burn-in samples to generate reasonable results. As a practical alternative, we experiment with greedy optimization from a random graph by greedily flipping edges so as to minimize the energy function. This will seek out local modes of the distribution, and empirically yields good results. This is also reminiscent of the peturb-and-MAP framework (Papandreou & Yuille, 2011), which generates exact samples from an approximate distribution that closely resembles the trained EBM.

We can also combine sampling strategies: initialize the chain via optimization, and then perform Gibbs sampling. This allows us to trade-off computation time and fidelity.

If we are working in a domain with known constraints on the generation process, such as molecular data, we can also use Metropolis-Hastings to incorporate these constraints into the proposal distribution, similar to (Seff et al., 2019).

4. Experiments

4.1. Setup

Baselines. We train our models on two graph datasets introduced in You et al. (2018b):

Ego-small. 200 graphs with $4 \leq |V| \leq 18$, drawn from the larger Citeseer network dataset (Sen et al., 2008).

Community-small. 100 2-community graphs that were generated procedurally, with $12 \leq |V| \leq 20$.

We compare with 3 recent, state-of-the-art baselines: GraphRNN (You et al., 2018b), Graph Normalizing Flows (GNF) (Liu et al., 2019), and EDP-GNN (Niu et al., 2020). We follow the same dataset train-test splits as the GraphRNN codebase (You et al., 2018a).

Evaluation. We use the evaluation script from You et al. (2018a), which computes MMD scores between generated samples and the test set for 3 graph statistics: degree, orbit, and cluster.

Training. We train a GNN-EBM for each dataset using three different estimators: pseudolikelihood, CNCE, and PCD. We use the same GNN architecture throughout, which consists of attention for each MP step, mean global pooling, and finally an MLP that outputs a single scalar. For CNCE, we tune the Beta noise distribution. For PCD, we tune the number of Gibbs steps needed to approximate a sample from the model.

For the initial node features, we compared Gaussian, one-hot, random binary, and Laplacian features and found that one-hot worked the best. We use these for all experiments.

Generation. We use the Erdos-Renyi model (Erdős & Rényi, 1960) with $p = 0.1$ as our initial distribution. We do greedy generation until the model reaches a local minimum, which typically takes 30-70 steps. We then do Gibbs sampling and find 1000 "mini-steps" is enough to generate high-quality samples. Each mini-step involves sampling the conditional for a single edge, as defined in Algorithm 2.

For more details on the setup and how we tuned hyperparameters, see Section 7.2.

4.2. Results

We include Figures 2-5 referenced in this section in the supplementary material, Section 7.

Generated Samples. We show samples in Figure 3.

Quantitative Evaluation. In Table 1, we compare the MMD scores for GNN-EBM trained with the different estimators. Pseudolikelihood and CNCE give competitive results on Ego-small but fail on the more complicated Community-small dataset. PCD is the clear winner, performing competitively against all baselines on both datasets.

Generalization. We investigated the generalization of the PCD-trained EBM in Figure 1. For each Community-small graph, we drew 100 permutations for its initial one-hot node features and plotted the average of the corresponding 100 negative energies. The train and test energies are overlapping and indistinguishable. We provide a similar plot for Ego-small in Figure 2. The same trend generally holds as well; the large test graph appearing at the bottom of the plot is the exception. We suspect the model struggles to generalize on larger graphs because there are so few of them in the Ego-small dataset.

We note an interesting trend: the model assigns higher negative energies to larger graphs. We will continue investigating

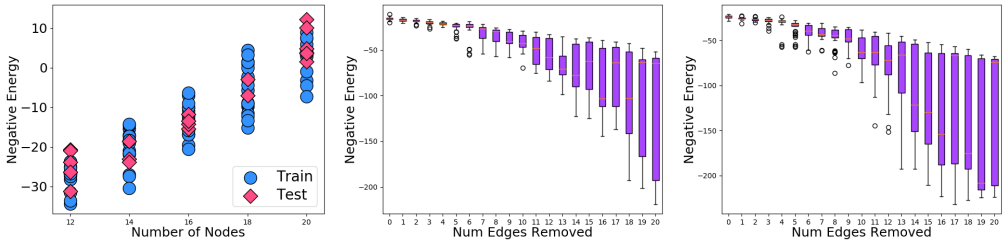

Figure 1. Left: For each Community-small graph we drew 100 random permutations of its initial node features, computed the corresponding 100 negative energies, and plotted the average as a single point. Datapoints sorted by number of nodes. Middle, Right: negative energies for random perturbations of a single Community-small test graph. Each column corresponds to 100 random perturbations.

Table 1. MMD scores for various graph statistics between 1024 graphs generated from each model and the test set. The scores for GRAPHRNN and GNF are copied directly from Liu et al. (2019). The scores for EDP-GNN are copied directly from Niu et al. (2020).

	COMMUNITY-SMALL			EGO-SMALL		
MODEL	DEGREE	CLUSTER	ORBIT	DEGREE	CLUSTER	ORBIT
GRAPHRNN	0.030	**0.010**	**0.010**	0.040	0.050	0.060
GNF	0.120	0.150	0.020	**0.010**	0.030	**0.0008**
EDP-GNN	**0.006**	0.127	0.018	**0.010**	**0.025**	0.003
PSEUDOLIKELIHOOD	0.390	0.630	0.335	0.030	0.065	0.014
CNCE	0.187	0.567	0.245	0.021	0.043	0.007
PCD	0.039	0.082	0.016	0.023	0.027	0.006

this and whether a different architecture or normalization scheme would help.

While pseudolikelihood and CNCE fail to produce high-quality samples for Community-small, we note that our models still generalize well over the test data and have included plots illustrating this in Figure 5. This suggests that the local energy landscape around data is well-formed.

Energy of Graph Perturbations. In Figure 1, we investigate the energy landscape around the dataset graphs. Taking a single test graph from Community-small, we generate random perturbations by adding or removing an increasing number of edges and observe that the negative energy decreases smoothly.

Permutation Invariance. While any GNN that uses the same MP function over all nodes is naturally permutation equivariant, it is not necessarily permutation *invariant*. If we generate one-hot node features and randomly assign them to a graph's nodes, the GNN will not necessarily map to the same output for each random assignment. However, we hope that our GNN would still learn some notion of permutation invariance. We investigate this in Figure 4, where we show the negative energy variance for different random node feature assignments for each dataset graph. Each column corresponds to a graph, and we sort by increasing number of nodes from left to right within the two partitions. While there are some outliers, the set of each graph's negative energies typically displays low variance and is outweighed by the variance between different graphs. The plots suggest our model has learned to assign similar energies to different assignments of the initial node features, with the variance increasing as the number of nodes (and thus, number of possible permutations) of a graph increases. We note that for Ego-small, there is much higher variance for larger graphs because there are so few of them in the dataset.

5. Conclusion

In this work we explored the use of GNN-EBMs for modeling distributions of graphs, applying various training techniques to produce a generative model which performs comparably to recent baselines.

Clear directions for future work include scaling our approach to larger graphs (You et al., 2018b), which presents interesting scalability challenges. Additionally, the energy-based parameterization of our model opens up a number of interesting applications. As in Grathwohl et al. (2019) we can re-purpose classification architectures to define EBMs, thus leveraging state-of-the-art graph classification models for challenging tasks such as molecular property prediction (Schütt et al., 2018; Klicpera et al., 2020) and train them as generative models, enabling intuitive approaches for semi-supervised classification and outlier detection.

6. Acknowledgements

We thank Aviral Kumar and William Chan for their helpful discussions and feedback.

References

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

Besag, J. Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 24(3):179–195, 1975.

Carreira-Perpinan, M. A. and Hinton, G. E. On contrastive divergence learning. In *Aistats*, volume 10, pp. 33–40. Citeseer, 2005.

Ceylan, C. and Gutmann, M. U. Conditional noise-contrastive estimation of unnormalised models. *arXiv preprint arXiv:1806.03664*, 2018.

Du, Y. and Mordatch, I. Implicit generation and modeling with energy based models. In *Advances in Neural Information Processing Systems*, pp. 3603–3613, 2019.

Erdős, P. and Rényi, A. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Fischer, A. and Igel, C. Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1): 25–39, 2014.

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.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., Norouzi, M., and Swersky, K. Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*, 2019.

Gutmann, M. and Hyvärinen, A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010.

Hinton, G., Osindero, S., Welling, M., and Teh, Y.-W. Unsupervised discovery of nonlinear structure using contrastive backpropagation. *Cognitive science*, 30(4):725–731, 2006a.

Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, 18 (7):1527–1554, 2006b.

Klicpera, J., Groß, J., and Günnemann, S. Directional message passing for molecular graphs. *arXiv preprint arXiv:2003.03123*, 2020.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. In *Advances in Neural Information Processing Systems*, pp. 13556–13566, 2019.

Niu, C., Song, Y., Song, J., Zhao, S., Grover, A., and Ermon, S. Permutation invariant graph generation via score-based generative modeling. *Artificial Intelligence and Statistics*, 2020.

Papandreou, G. and Yuille, A. L. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *2011 International Conference on Computer Vision*, pp. 193–200. IEEE, 2011.

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, 2008.

Schütt, K. T., Sauceda, H. E., Kindermans, P.-J., Tkatchenko, A., and Müller, K.-R. Schnet–a deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24):241722, 2018.

Seff, A., Zhou, W., Damani, F., Doyle, A., and Adams, R. P. Discrete object generation with reversible inductive construction. In *Advances in Neural Information Processing Systems*, pp. 10353–10363, 2019.

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

Simonovsky, M. and Komodakis, N. GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pp. 412–422. Springer, 2018.

Tieleman, T. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pp. 1064–1071, 2008.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

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

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *Knowledge Discovery & Data Mining*, pp. 974–983, 2018.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. Code for GraphRNN: Generating realistic graphs with deep auto-regressive model. https://github.com/JiaxuanYou/graph-generation, 2018a.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018b.

Younes, L. Parametric inference for imperfectly observed gibbsian fields. *Probability theory and related fields*, 82 (4):625–645, 1989.

7. Supplementary Material

7.1. Algorithms

Algorithm 1 CNCE Corrupted Graph Generation

Input: graph $\mathcal{G}$ with $|\mathcal{V}|$ nodes
Initialize $A = \text{adjacency}(\mathcal{G})$.
Sample $p \sim \text{Beta}(\alpha, \beta)$
for $i = 1$ **to** $|\mathcal{V}|$ **do**
  for $j = i + 1$ **to** $|\mathcal{V}|$ **do**
    Sample $U \sim Unif(0, 1)$
    if $U < p$ **then**
      $A \leftarrow A_{\overline{ij}}$
    end if
  end for
end for
return $\text{graph}(A)$ // Create a new corrupted graph from A.

7.2. Training Details

GNN Architecture. We use 6 MP steps. Each MP step consists of dot-product attention (Vaswani et al., 2017) as the aggregation function and an MLP with 3 layers and 1024 hidden units per layer. We follow this with mean global pooling to generate a graph-level embedding. We feed this into an MLP with 3 layers and 1024 hidden units, which outputs our negative energies.

Hyperparameter Search. For the CNCE corrupted graph generation, we searched over the following parameters for the Beta distribution: $\{(\alpha = 1, \beta = 10), (\alpha = 1, \beta = 20), (\alpha = 2, \beta = 8)\}$. $(\alpha = 1, \beta = 20)$ worked the best.

For the number of Gibbs "mini-steps" in PCD training, we searched over $N = \{25, 50, 100, 200\}$. 100 was sufficient, 50 was not enough.

Following You et al. (2018b), we tune parameters by generating 1024 samples from the model, computing the MMD scores between these samples and the train graphs, and picking the parameters that result in the best score, averaged across the three statistics.

Algorithm 2 Gibbs Sampling

Input: initial graph $\mathcal{G}$ with $|\mathcal{V}|$ nodes, Gibbs steps N
Initialize $A = \text{adjacency}(\mathcal{G})$
for $i = 1$ **to** N **do**
  // One full step.
  for $j = 1$ **to** $|\mathcal{V}|$ **do**
    for $k = j + 1$ **to** $|\mathcal{V}|$ **do**
      // One "ministep".
      $A' \leftarrow A_{\overline{jk}}$
      $p \leftarrow \frac{e^{f(A;\theta)}}{e^{f(A;\theta)} + e^{f(A';\theta)}}$
      Sample $U \sim Unif(0, 1)$
      if $U > p$ **then**
        $A \leftarrow A'$
      end if
    end for
  end for
end for
return $\text{graph}(A)$

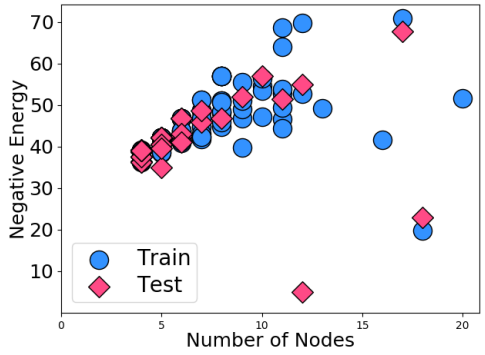

Figure 2. For each graph in Ego-small we drew 100 random permutations for its initial one-hot node features, computed the corresponding 100 negative energies using our PCD-trained EBM, and plotted the average as a single point. Points sorted by number of nodes in the graph.

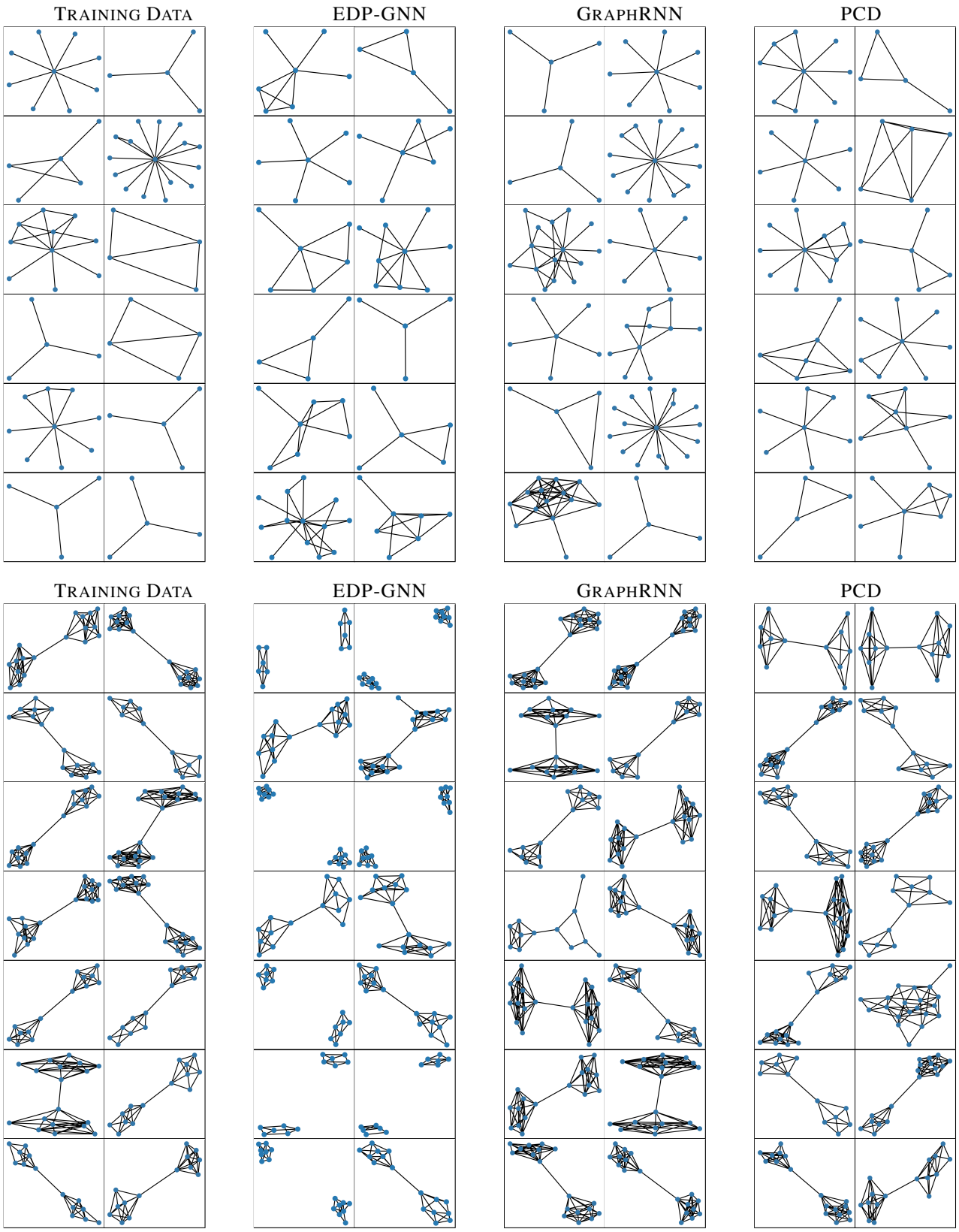

Figure 3. Graph samples. Top: Ego-small, bottom: Community-small. EDP-GNN samples are copied from Niu et al. (2020), and GraphRNN samples are generated by training a model using their provided code (You et al., 2018a).

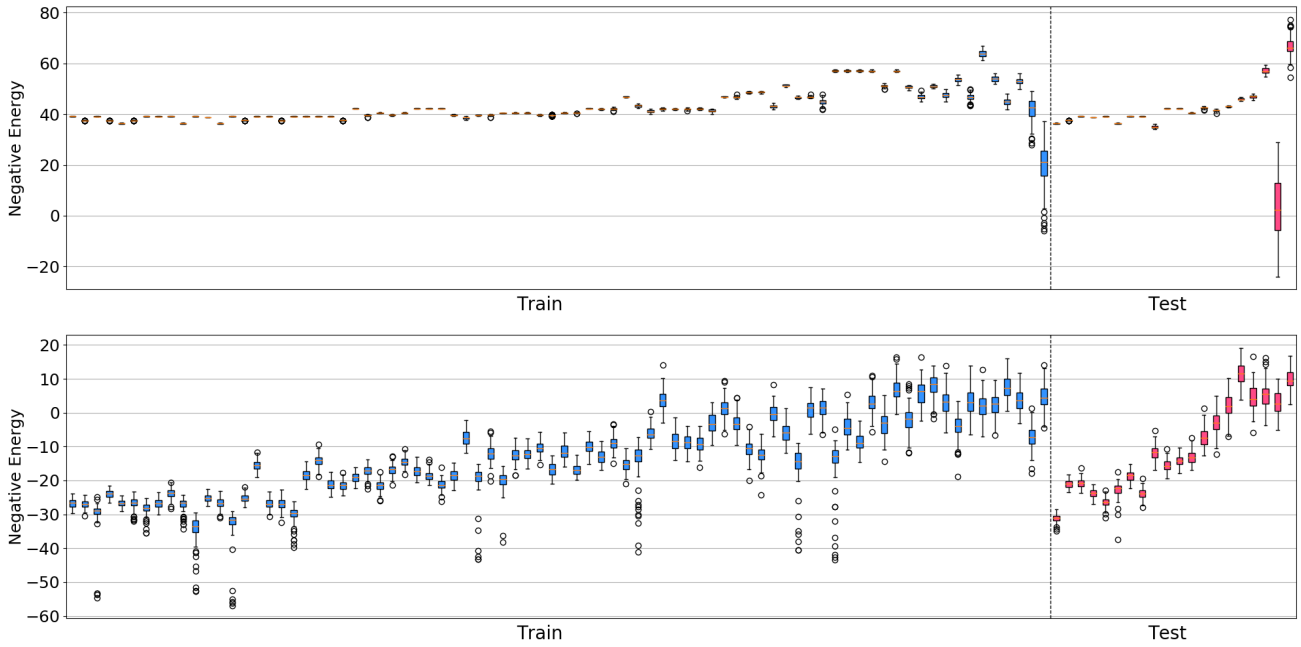

Figure 4. Top: Ego-small. Bottom: Community-small. Boxplots depicting variance in negative energy over permutations of each dataset graph. For each graph, we drew 100 random permutations for its initial one-hot node features and plotted the corresponding 100 negative energies from the PCD-trained EBM in a single column. Within Train and Test, graphs are sorted by increasing number of nodes from left to right. We randomly subsampled Ego-small to make the plot more readable.

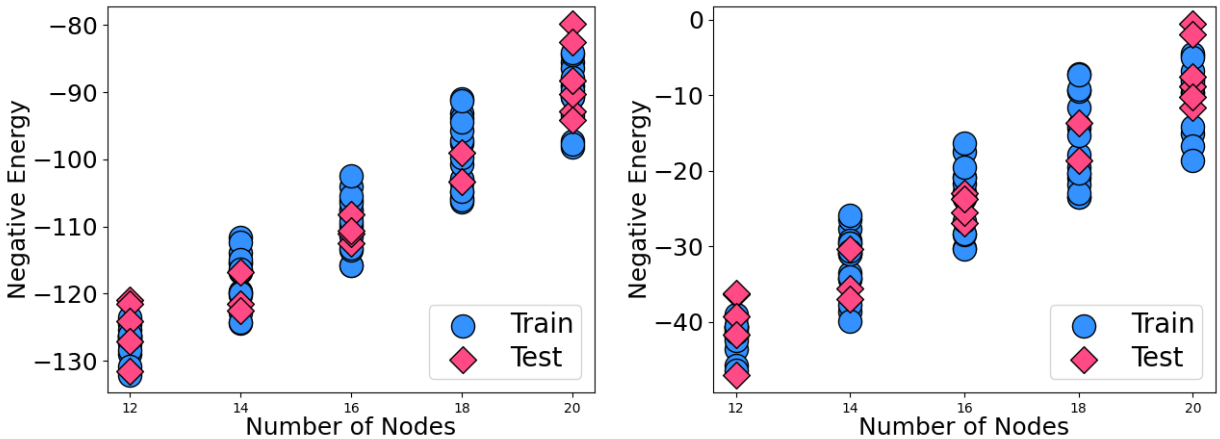

Figure 5. Left: Pseudolikelihood-trained EBM. Right: CNCE-trained EBM. For each dataset graph in Community-small we draw 100 random permutations for its initial one-hot node features, compute the corresponding 100 negative energies using our EBM, and plot the average as a single point. Points sorted by number of nodes in the graph.