Graph Generation with Graphon Generative Adversarial Networks

Peter Jung[1] and Ondřej Kuželka[1]

FEE, Czech Technical University, Prague, Czech Republic https://fel.cvut.cz/en/

Abstract. Graphons are limits of converging sequences of graphs with a particularly simple representation—a graphon is simply a symmetric function of two variables on $[0;1]^2$. In this work, we develop an elegant GAN model, called GraphonGAN, which uses graphons implemented by neural networks as generators and graph neural networks as discriminators. We show that GraphonGAN is a decent model for modelling real-world networks. All the source codes will be available at https://github.com/kongzii/deepgraphon.

Keywords: Graph · Generation · Graphon · GAN.

1 Introduction

Modelling graphs is a fundamental problem with applications in many different fields and learning to generate random graphs from examples is a challenging task. In this work, we propose a new method, particularly well suited for the efficient generation of dense and not-so-dense networks.

1.1 Related Work

One of the first attempts to model graphons using neural networks was made in [1], however, that work was limited to simple artificial graphons that were known in advance and were not tested on real-world graphs. The work [2] used auto-encoders to learn graphon representation from underlying graphs, but was evaluated only on downstream classification tasks. Other papers took advantage of graphon properties. For instance, [3] used graphons to initialize weights of a GNN for later classification tasks. In [4], artificial graphons were learnt via Structured Gromov-Wasserstein Barycenters and although the authors do not compare properties of generated graphs, they do compare learned graphon with their ground truth representation, but this is not a good way to evaluate learned graphons because two graphons can represent the very same distribution over graphs although they look visually different because of isomorphism. In [5], GNNs were trained by resampling from the graphon, which is however estimated on the whole dataset and the work itself did not focus on learning graphons from graphs.

2 Method

TL; DR In a nutshell, our model is a GAN that uses graphons, implemented as neural networks, as generators and GNNs as discriminators.

2.1 Generator

The generator is a model representing the graphon, whose task is to learn how to generate realistically-looking graphs with respect to the training ones. Since a graphon is nothing else than a symmetric function from $[0;1]^2$ to $[0;1]$, we can use a neural network with two inputs and one output to represent it. To sample a graph on n vertices from the generator, the basic idea would be the same as when sampling graphs from any graphon. We would sample n points x_1, x_2, ..., x_n from the uniform distribution $U(0,1)$. For every pair (x_i, x_j), where $i < j$, we would sample a value y_{ij} from the Bernoulli distribution with success probability equal to $W(x_i, x_j; \mathbf{w})$, where $W(x_i, x_j; \mathbf{w})$ is the graphon function represented by the neural net with weights $\mathbf{w}$. The adjacency matrix of the sampled graph would then be the matrix $(y_{ij})+(y_{ij})^T$. In other words, the sampled graph would contain an edge between vertices i and j if and only if $y_{ij} = 1$. This corresponds exactly to how graphs are sampled from graphons, however, if we implemented the model exactly like this, we would not be able to propagate gradients through it due to the use of the discrete Bernoulli distributions. To allow propagating gradients through our architecture, we need to use the Gumbel-Softmax trick [6]. Therefore we use a modified discrete sampling procedure, where the Gumbel-Softmax trick is used to sample a discrete adjacency matrix that is differentiable in the backward pass (we omit details due to limited space).

2.2 Discriminator

The discriminator in our architecture is a standard GNN, which is trained to discriminate real graphs from the training set from graphs generated by the generator (graphon). It receives adjacency matrices of the graphs.

2.3 Training

Every epoch consists of two stages. First, during the generator training, the neural network, which represents the graphon (generator), generates a sample of the approximate adjacency matrix $(\tilde{y}_{ij})$. This sample is fed to the discriminator, which has fixed weights when training the generator. We then take the gradient steps trying to maximize the log-loss of the discriminator (the generator wants to fool the discriminator)—the reason we can do this is the fact that due to the use of the Gumbel-Softmax, the architecture is differentiable. Second, during the discriminator training, the graphon (generator) generates an artificial graph, but this time we fix the weights of the generator and pass the generated graph to the discriminator as a fake sample to learn on. In this stage, a real graph from

the training set is passed to the discriminator with the true-sample label. For the discriminator training, we take gradient steps minimizing the log-loss of the discriminator on this pair of samples.

3 Experiments

3.1 Generating networks

Evaluation Metrics. Evaluating generative graph models is not straightforward. Recently, this problem was studied in depth by [7], who observed that there is no single universal similarity metric that would work reliably across all graphs. As a solution, the authors of [7] proposed five different metrics based on topological data analysis (TDA), specifically on so-called Betti curves. They also proposed a method how to choose the best metric. They suggested executing multiple graph perturbations with increasing probability, and to choose the metric that is closest to being monotonically increasing with the increasing perturbation.

Datasets. We use the protein-protein association networks of Abiotrophia and Saccharomyces from the STRING database [8] and seven different graph datasets containing graphs with different properties: small, sparse, big or dense, these datasets are available via TUDataset [9]. In addition, we evaluate also graphs generated from artificial graphons. The smallest graph family has on average 24.9 nodes and 77.48 edges, the biggest one has 6394 nodes and 994296 edges, we took graphs of different sizes to prove that our method is scalable and usable across a variety of graphs.

Evaluation on Single-Graph Datasets. When evaluating single-graph datasets, one needs to be careful to make the evaluation meaningful. For training, we use an induced subgraph obtained by sampling half of the vertices without replacement and then we test the learned model on the graph induced on the remaining vertices.

Evaluation on Multiple-Graph Datasets. Evaluating generative graph models on datasets consisting of multiple graphs is straightforward. We split the dataset into the training set, containing 50% of graphs from the dataset, and the test set, containing the rest.

Results. We compared our method with two other deep learning-based graph generative methods [10, 11] and two non-neural network based methods (GWB, SGWB, [4]). GraphonGAN won on 17 from 21 cases, GraphRNN won 1 time, GRAN 1 time and GWB 2 times.

4 Conclusions

We introduced a new graph generative model. The model achieves good experimental results on real and artificial datasets. One of the main benefits of our

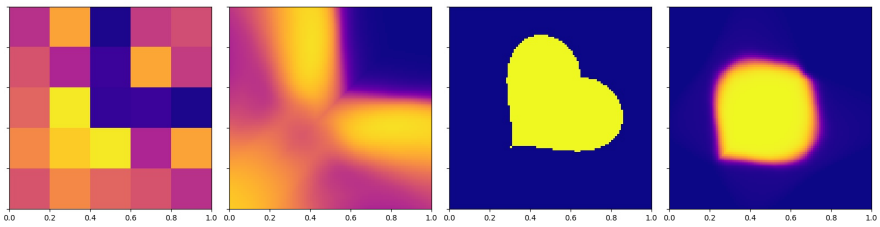

Fig. 1. Artificial graphon (left) and its learned counter-part (right).

methods is its efficiency and simplicity. One set of hyper-parameters was used for all the experiments, meaning the approach yields universally good results. Lastly, graphon can be sampled in a full parallel fashion, meaning this method can potentially scale to arbitrary large graphs on both training and inference. Theoretical real-world use cases of our method include unsupervised pre-training, clustering or enhancement of training datasets.

References

1. Huy Hoang Vu: Learning Graphons Using Neural Networks (2020)
2. Hongteng Xu, Peilin Zhao, et al.: Learning Graphon Autoencoders for Generative Graph Modeling, https://arxiv.org/pdf/2105.14244.pdf (2021)
3. Luana Ruiz, Luiz F. O. Chamon, et al.: Graphon Neural Networks and the Transferability of Graph Neural Networks, https://arxiv.org/pdf/2006.03548.pdf (2020)
4. Hongteng Xu1, Dixin Luo, et al.: Learning Graphons via Structured Gromov-Wasserstein Barycenters (2021)
5. Ziqing Hu, Yihao Fang, Lizhen Lin: Training Graph Neural Networks by Graphon Estimation, https://arxiv.org/pdf/2109.01918.pdf (2021)
6. Eric Jang, Shixiang Gu, et al.: Categorical Reparameterization with Gumbel-Softmax (2017)
7. Leslie O'Bray, Max Horn, et al.: Evaluation Metrics for Graph Generative Models: Problems, Pitfalls, https://arxiv.org/abs/2106.01098v1 (2021)
8. Szklarczyk, D and Gable, et al.: Protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets (2019)
9. Christopher Morris, Nils M. Kriege, et al.: TUDataset: A collection of benchmark datasets for learning with graphs (2020)
10. Liao, Renjie, et al.: Efficient Graph Generation with Graph Recurrent Attention (2019)
11. Jiaxuan You, Rex Ying, et al.: GraphRNN: A Deep Generative Model for Graphs, http://arxiv.org/abs/1802.08773 (2018)
12. Nicolò Vallarano, Matteo Bruno, et al.: Fast and scalable likelihood maximization for Exponential Random Graph Models (2021)