ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0.
Available from http://www.i6doc.com/en/.

Graph generation by sequential edge prediction

Davide Bacciu[1], Alessio Micheli[1], Marco Podda[1] *

[1] University of Pisa - Dept. of Computer Science
Largo Bruno Pontecorvo 3, Pisa - Italy

Abstract. Graph generation with Machine Learning models is a challenging problem with applications in various research fields. Here, we propose a recurrent Deep Learning based model to generate graphs by learning to predict their ordered edge sequence. Despite its simplicity, our experiments on a wide range of datasets show that our approach is able to generate graphs originating from very different distributions, outperforming canonical graph generative models from graph theory, and reaching performances comparable to the current state of the art on graph generation.

1 Introduction

Graphs allow to efficiently store and access relational data: their use is strategical to encode information in domains such as Bioinformatics, Cheminformatics, Natural Language Processing, and many more. Given their superior expressiveness with respect to "flat" vectorial data, many Machine Learning models accept graphs as inputs, to create richer and more informed predictors on a variety of tasks [1]. One learning problem of particular interest is how to instruct Machine Learning models to generate graphs from arbitrary distributions. This problem is inherently hard for a number of reasons: *a*) the space of graphs is exponential (for a fixed number of nodes N there are $2^{N(N-1)}$ possible undirected graphs), meaning that there are severe limitations in how much of it can be explored when searching for a solution; *b*) graphs are discrete objects, hence learning algorithms like back-propagation (which require the objective function to be continuously differentiable) are not directly applicable; *c*) aside from few restricted domains, there is often no clear-cut way to evaluate the quality of a sample with respect to the unknown graph distribution. In this paper, we propose an approach to address these challenges. Instead of working directly in graph space, we cast the generative process of a graph as a sequential one, where we learn to predict its *ordered edge sequence*. This has two benefits: by superimposing an order on the generative process, we restrict the number of possible outcomes and make the problem tractable; furthermore, moving to sequences domain allows us to employ reliable Recurrent Neural Network architectures (often included in the Deep Learning framework) for learning and generation. We implemented and tested our model in a wide range of heterogeneous graph datasets, outperforming classical baselines such as the Erdös-Rényi and the Barabási-Albert models (sometimes by several orders of magnitude), reaching competitive results with respect to the current state of the art on the generative task.

*This work has been supported by the Italian Ministry of Education, University, and Research (MIUR) under project SIR 2014 LIST-IT (grant n. RBSI14STDE).

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

2 Model

A graph is a pair $G = \langle V, E \rangle$, where $V = \{v_0, v_1, \ldots, v_n\}$ is the set of vertices (or nodes) and $E = \{(v, u) \mid v, u \in V\}$ is a set of edges (or links). We define $|V| = N$ and $|E| = M$. Let us assume that the graphs we deal with are fully connected, undirected, i.e. $(v, u) = (u, v)$, $\forall v, u \in V$, and do not contain self-loops, i.e. edges of the form (v, v). We furthermore assume the existence of a bijective labeling function $\pi : V \to \mathbb{N}_0$ that assigns a non-negative integer to each node in the graph following some ordering criteria. If π is given, we can represent a graph G with its *ordered edge sequence*, defined as:

$$S_G = s_1, \ldots, s_M, \text{ with } s_i = (x_i, y_i), \text{ where } (\pi^{-1}(x), \pi^{-1}(y)) \in E.$$

Moreover, we impose $s_i \leq s_{i+1}$ iff $x_i < x_{i+1}$ or $(x_i = x_{i+1}$ and $y_i < y_{i+1})$, i.e. S_G is sorted in ascending lexicographical order. Clearly, the ordered edge sequence of a graph is dependent on the particular choice of π; that is, different node orderings lead to different edge sequences. We discuss how π can be chosen in Section 3.

To generate a graph, one would usually learn its probability distribution $P(G)$ from data, and sample from it. Both learning and sampling are hard since graph are invariant to node permutations, which makes the space of graphs exponential. Instead, we aim at solving an alternative problem: learning the ordered edge sequence of a graph or, in other terms, maximizing the ordered edge sequence probabilities of the graphs in a dataset:

$$\max P(S_G) = \prod_{i=1}^{M} P(s_{i+1} \mid s_i).$$

In the above formula, S_G is the ordered edge sequence, s_i is its i-th element, and the two components of the pair (x, y) are the integers assigned to the nodes by π, that identify the generic edge in G. We propose to learn this probability with a sequence of two Recurrent Neural Networks: one learns $P(x_{i+1} \mid x_i)$, i.e. the sequence of the first elements of the pairs in S_G, and another that, given the latter sequence, learns $P(y_i \mid x_i)$ (the second element of the pair). Intuitively, the first Neural Network learns which node the next pair in the sequence should contain as its first component; its hidden state updates a latent representation of the neighborhood of the input node. The second network learns to predict adjacent nodes; its hidden state keeps track of the graph connectivity as nodes are added. We implemented the architecture of the two networks using Gated Recurrent Units [2] and softmax outputs (where the output classes are the identifiers of the nodes).

Figure 1 provides an insight on the inner workings of the model. During training, we start from a graph, whose nodes are first labelled according to the node ordering procedure. The second step consists in extracting the ordered edge sequence. The first components of the ordered edge sequence are fed to the first network, which predicts the next node identifier in the sequence (using

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0.
Available from http://www.i6doc.com/en/.

teacher forcing). The sequence output of the first network is then fed as input to the second network which, for each node identifier, predicts the identifier of its adjacent node. The whole architecture is trained to minimize the categorical cross-entropy between the ground truth and the desired network outputs.

The inference phase starts with the first network, whose softmax layers are sampled sequentially to produce a sequence of node identifiers. The generated sequence becomes the input for the second network, which again predicts all the needed node adjacencies. The outputs of the two networks are finally combined into a novel ordered edge sequence, from which the new graph can be reconstructed.

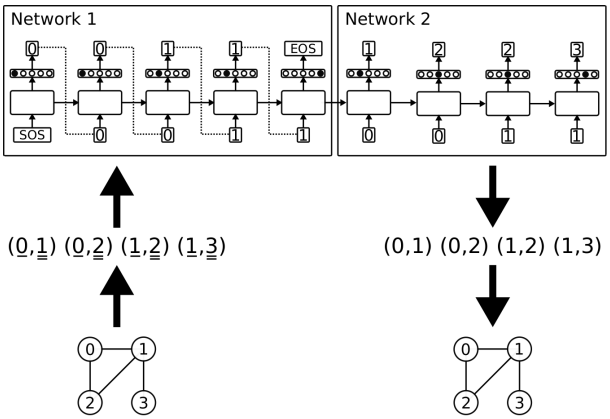

Fig. 1: Proposed model for learning graph generation. In the ordered edge sequence on the left, underlined nodes indicate the desired output for the first network; doubly underlined nodes indicate the desired output for the second network. Upward arrows describe data flow during training (start from a graph, end up with the network output). Downward arrows describe data flow during inference (start from the network, end up with a novel graph). Dashed lines indicate a sampling procedure (used only during inference).

3 Experiments

Following, we review our workflow to evaluate the goodness of our architecture: in particular, we briefly describe the datasets used for learning, the baselines we compared to, the metric of choice to evaluate the goodness of the samples and some details of the experimental setup.

Datasets: our model was trained on a set of five heterogeneous graph datasets. Each one represents different graph distributions that model very complex node/edges dependencies: Ladders and Community are synthetic datasets of ladder graphs and two-community (i.e. graphs composed of two densely connected components weakly connected among themselves) respectively; Ego is a

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

Name	# of graphs	Min. nodes	Max. nodes	Avg. nodes	Avg. edges	Training/Test split
Ladders	45	10	98	54.00	79.00	45/45
Community	200	16	24	19.99	55.85	150/50
Ego	200	16	32	22.45	31.41	150/50
Enzymes [4]	350	15	40	27.96	55.47	200/150
Proteins [4]	500	15	40	26.17	49.90	350/150

Table 1: Statistics of datasets used in the experiments.

subset of ego-networks extracted from the Citeseer dataset [3]; Enzymes and Proteins are subsets of real-world graph datasets representing enzymes and proteins respectively. Dataset statistics are presented in Table 1.

Choice of π: as said before, the choice of the ordered labelling function π is crucial, because it defines how the ordered edge sequence is generated. However, while a notion of ordering can be assumed for certain types of graphs such as molecules (where nodes can be labelled in sequential order following the SMILES [5] encoding), finding such a function is challenging for graphs like social networks, which have no clear notion of "what comes first". Thus, for graphs in the Ladders, Ego and Community datasets, we employed the following procedure: *a*) take one node v at random in the graph; *b*) construct a BFS tree of the graph rooted at v; *c*) label the nodes in the graph according to the order of visit of the tree. While this algorithm effectively creates orderings that are not consistent across graphs, we found no observable impact in terms of performances in our experiments. However, we are aware that this can be a bias, whose effects can be further investigated and eventually relaxed in future works.

Baselines: to get a broader sense on how well our model performs, we compared its performances to a total of three baseline models. Two of them are classical generative models of graphs coming from graph theory literature: the Erdös-Rényi [6] (E-R) and the Barabási-Albert [7] (B-A) models. We also compared to a Deep Learning based approach named GraphRNN (GRNN) model developed by You et al. [8]. In early experiments, we also evaluated other Deep Learning based approaches such as [9] and [10], which however did not give satisfactory results and were thus discarded.

Metrics: the quality of the generated graphs was assessed by computing the Kullback-Leibler Divergence [11] (KL-D) of graph statistics distributions between graphs in the test set and graphs generated by the models. The statistics of choice were degree distribution (DD) and clustering coefficient (CC). Since the KL-D is defined for probability distributions, we constructed a probability vector for each of the two statistics using the following procedure: *a*) for each graph, compute its statistics vector; *b*) 0-pad the vector to reach the size of the largest statistics vector in the dataset; *c*) sum up each padded vector and normalize. Since the CC is continuous, we discretized it using a histogram with 100 bins.

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

	Ladders		Community		Ego	
Model	DD	CC	DD	CC	DD	CC
E-R	1.1414	**0.0000**	0.0480	0.9827	0.4997	0.9896
B-A	1.3082	**0.0000**	0.2352	0.4050	0.0766	0.4995
GRNN	0.0148	0.0001	**0.0113**	**0.0463**	0.0598	0.1767
Ours	**0.0092**	**0.0000**	0.0115	0.0815	**0.0381**	**0.1071**

	Enzymes		Proteins	
Model	DD	CC	DD	CC
E-R	0.2352	1.2527	0.3089	1.484
B-A	0.5839	0.8423	0.7523	1.2099
GRNN	0.0410	0.0604	**0.0176**	**0.0254**
Ours	**0.0220**	**0.0222**	0.0450	0.0535

Table 2: KL-divergence between samples generated by the models and samples in the test set, evaluated on their degree distribution (DD) and clustering coefficient (CC). Best performances are in bold.

Experimental Setup: data was split between training and test as described in Table 1. At inference time, we sampled 1000 graphs from each trained model, and use them with the test set to calculate the corresponding performance metric. The GRNN model used 3 hidden layers with 64 recurrent units each, while our model used 2 hidden layers with 128 units each. Both models were trained for 1000 epochs, and we report negligible differences in terms of training time (for the Enzymes dataset, GRNN training took 2h.55m approximately, while training our model took 2h.45m approximately). Optimization was carried out with the Adam [12] optimizer, setting the learning rate to 0.003 for GRNN (with learning rate annealing) and 0.001 for our model. The parameters of the E-R and B-A models were estimated through maximum likelihood.

4 Results and Discussion

Table 2 shows the results of our experiments: our model obtains the best KL-D in three out of five datasets, and performs comparably to the GRNN model in the remaining two. It can be seen how both our model and the GRNN significantly outperform classical generative models in every dataset, sometimes (e.g. Ladders DD, Enzymes CC) by several orders of magnitude. Interestingly, for each dataset the best performing models score better in both statistics under consideration. Poor performances of the E-R and B-A models were expected since they are not designed to model real-world graphs, and are able to maintain only predefined properties during generation. Both our model and GRNN, instead, learn the graph distribution of interest adaptively, providing the necessary flexibility to

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

capture complex dependencies directly from data. Still, there is work to be done, in particular to remove possible biases in both how the learning phase is conceived and how the generation is carried on. In conclusion, we developed a conceptually simple, yet powerful, generative model for graph generation which approximates its ordered edge sequence via Recurrent Neural Networks, which is capable of reaching performances comparable to state of the art in the task.

References

[1] Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual graph markov model: A deep and generative approach to graph processing. In *Proc. of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[2] Kyunghyun Cho, Bart van Merriënboer, ÇaÄlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP '14, pages 1724–1734. Association for Computational Linguistics, 2014.

[3] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. CiteSeer: An Automatic Citation Indexing System. In *Proceedings of the Third ACM Conference on Digital Libraries*, DL '98, pages 89–98. ACM, 1998.

[4] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. BRENDA, the enzyme database: Updates and major new developments. *Nucleic acids research*, 32:D431–3, 2004.

[5] David Weininger, Arthur Weininger, and Joseph L. Weininger. SMILES. 2. Algorithm for generation of unique SMILES notation. *Journal of Chemical Information and Computer Sciences*, 29(2):97–101, 1989.

[6] P. Erdös and A. Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[7] Albert-László Barabási and Réka Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.

[8] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 5694–5703. JMLR.org, 2018.

[9] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning Deep Generative Models of Graphs. *CoRR*, abs/1803.03324, 2018.

[10] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning â ICANN 2018 â Part 1*, volume 11139 of *Lecture Notes in Computer Science*, pages 412–422. Springer, 2018.

[11] Solomon Kullback and Richard A. Leibler. On Information and Sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 1951.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, ICLR '15, 2015.