Graph Deep Generative Models

Octavian-Mihai Radu
AI Multimedia Lab
National University of Science and Technology
POLITEHNICA Bucharest
Bucharest, Romania
ORCID: 0009-0002-1634-5065

Bogdan Ionescu
AI Multimedia Lab
National University of Science and Technology
POLITEHNICA Bucharest
Bucharest, Romania
bogdan.ionescu@upb.ro

***Abstract*—Graphs are essential representations of complex systems across diverse fields like biology, engineering, and social science. However, creating models that accurately capture the intricate structure of real-world graphs and generate new, realistic graphs is a significant challenge. This difficulty stems from the unique characteristics of graphs: their high dimensionality, the non-unique ways they can be represented, and the intricate dependencies between their edges. These factors make it difficult to define and sample from probability distributions over graph space. Despite all the complexities associated with graph representation and generation, Graph Deep Generative Models have emerged as a promising tool for effectively learning and synthesizing graph structures. This work subsequently introduces the Graph Convolutional Policy Network (GCPN), a novel framework leveraging graph convolutional networks within a reinforcement learning paradigm for goal-directed graph generation. This approach is particularly relevant to molecular graph generation, where the objective is to design novel molecules exhibiting specific properties (e.g., drug-likeness, synthetic accessibility) while adhering to fundamental chemical constraints (e.g., valency).**

***Index Terms*—Graph Neural Networks (GNN), Recurrent Neural Networks(RNN), autoregressive models, molecular design, drug discovery**

I. Introduction and Related Work

The development of generative graph models' history has numerous methods already proposed for generating graphs based on predefined structural assumptions. However, a key challenge still remains: developing methods capable of directly learning generative models from a set of observed graphs. Learning generative models directly from data is an important step towards improving the fidelity of generated graphs. This capability also paves the way for new applications, such as discovering novel graph structures. Furthermore, it allows for completing evolving graphs by learning the underlying generative process. In contrast, traditional generative models for graphs are typically hand-engineered. These models are designed to simulate specific families of graphs. Thus, they lack the capacity to learn a generative model directly from observed data.

Traditional generative models for graphs, in contrast, are hand-engineered for specific graph families, limiting their ability to learn generative patterns directly from observed data. Deep generative models, including VAEs [3] and GANs [4], have significantly improved the ability to create realistic and complex data, such as images and text. A fundamental challenge in drug discovery and materials science lies in designing molecules that possess specific desired properties. This task is incredibly complex due to the large size of chemical space. In this paper we will also describe Graph Convolutional Policy Network (GCPN), an approach to generate molecules where the generation process can be guided towards specified desired objectives, while restricting the output space based on underlying chemical rules.

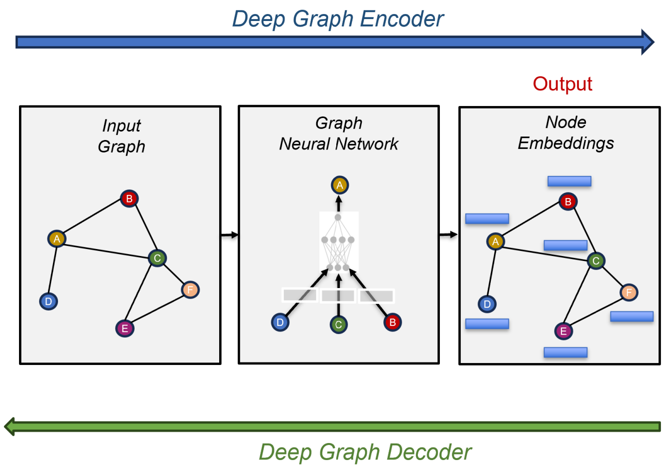


Fig. 1: Encoder-Decoder Graph Generation.

II. Deep Generative Models Overview

A. Generative Models

In general, for data generation (or in this case the graph generation) we assume that data are samples from $p_{data}(G)$ distribution. Therefore, p_{data} is the data distribution, which is never known to us, but we have sampled $G_i \sim p_{data}(G)$ from it. Our goal in generative modeling is to learn a $p_{model}(G;\theta)$ distribution, parametrized by θ, that we use to approximate $p_{data}(G)$. After that the generation process is just sampling from p_{model}. The most common approach to sample from a complex distribution is to:

- sample first from a simple noise distribution, like $z_i \sim N(0,1)$
- transform the noise via $x_i = f(z_i;\theta)$

1st Workshop on Artificial Intelligence for Multimedia,
AI Multimedia Lab, CAMPUS Research Institute, POLITEHNICA Bucharest,
Bucharest, Romania, 8 November, 2024

where $f(.)$ is a deep neural network trained from the data we have.

Most known architectures like Variational Auto Encoders (VAEs) or Generative Adversial Nets (GANs) have two models one for density estimation and one for sampling. In Figure 1.

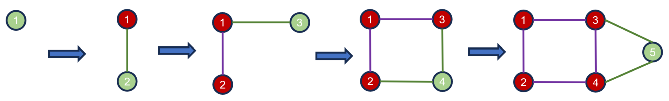


Fig. 2: Auto-Regressive Graph Generation.

are represented a Graph Encoder as density estimation and a Graph Decoder as sampling.

B. *Auto-regressive Generative Models*

In the case of auto-regressive models $p_{model}(G;\theta)$ is used for both density estimation and sampling. The general main idea of auto-regressive modeling is to express the joint distribution as a product of conditional distributions (chain rule):

$$p_{model}(G;\theta) = \prod_{t=1}^{n} p_{model}(g_t|g_1, ...g_{t-1};\theta)$$

In the graph generative modeling g_t will be the t-th action (add node, add edge).

An auto-regressive graph generation example is shown in figure 2.

III. Graph Recurrent Neural Networks

Recurrent neural networks are part of the auto-regressive family and widely used for data generation. In the case of the graphs, one such approach is described in "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models" [5]. I will briefly describe it how it works.

An undirected graph $G = (V, E)$ is defined by its node set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{(v_i, v_j)|v_i, v_j \in V\}$. One common way to represent a graph is using an adjacency matrix, which requires a node ordering π that maps nodes to rows/columns of the adjacency matrix. More precisely, π is a permutation function over V.

GRNN first define a mapping f_S from graphs to sequences, where for a graph $G \sim p(G)$ with n nodes under node ordering π:

$$S^\pi = f_S(G, \pi) = (S_1^\pi, ..., S_n^\pi), \tag{1}$$

where each element $S_i^\pi \in \{0,1\}^{i-1}, i \in \{1, ..., n\}$ is an adjacency vector representing the edges between node $\pi(v_i)$ and the previous nodes $\pi(v_j), j \in \{1, ..., i-1\}$ already in the graph:

$$S_i^\pi = (A_{1,i}^\pi, ..., A_{i-1,i}^\pi)^T, \forall i \in \{2, ..., n\}. \tag{2}$$

For undirected graphs, S^π determines a unique graph G, and GRNN write the mapping as $f_G(\cdot)$ where $f_G(S^\pi) = G$.

Thus, instead of learning $p_{model}(G)$, whose sample space cannot be easily characterized, GRNN sample the auxiliary π to get the observations of S^π and learn $p_{model}(S^\pi)$, which can be modeled auto regressively due to the sequential nature of S^π. At inference time, GRNN can sample G without explicitly computing $p_{model}(G)$ by sampling S^π, which maps to G via f_G.

Given the above definitions, we can write $p(G)$ as the marginal distribution of the joint distribution $p(G, S^\pi)$:

$$p_{model}(G) = \sum_{S^\pi} p(S^\pi) I[f_G(S^\pi) = G], \tag{3}$$

where $p(S^\pi)$ is the distribution that we want to learn using a generative model. Due to the sequential nature of S^π, we further decompose $p(S^\pi)$ as the product of conditional distributions over the elements:

$$p(S^\pi) = \prod_{i=1}^{n+1} p(S_i^\pi|S_1^\pi, ..., S_{i-1}^\pi) \tag{4}$$

where we set S_{n+1}^π as the end of sequence token EOS, to represent sequences with variable lengths. We simplify $p(S_i^\pi|S_1^\pi, ..., S_{i-1}^\pi)$ as $p(S_i^\pi|S_{<i}^\pi)$ in further discussions.

The algorithm proposed in [5] is summarized below.

GraphRNN inference algorithm

```
Input: RNN-based transition module $f_{trans}$, output module $f_{out}$, probability distribution $\mathcal{P}_{\theta_i}$ parameterized by $\theta_i$, start token SOS, end token EOS, empty graph state $h'$
Output: Graph sequence $S^\pi$
$S_1^\pi = \texttt{SOS}$, $h_1 = h'$, $i = 1$
repeat
    $i = i + 1$
    $h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$ {update graph state}
    $\theta_i = f_{\text{out}}(h_i)$
    $S_i^\pi \sim \mathcal{P}_{\theta_i}$ {sample node $i$'s edge connections}
until $S_i^\pi$ is EOS
Return $S^\pi = (S_1^\pi, ..., S_i^\pi)$
```

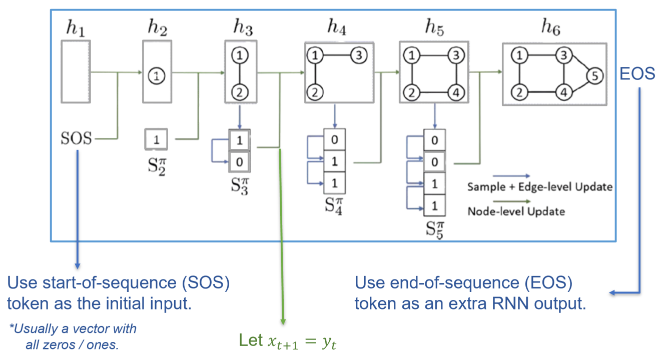
The data flow diagram that corresponds to the GraphRNN inference algorithm is in Figure 3. It can be observed the tow recurrent neural networks: the node RNN and the edge RNN. Start of Sequence (EOS) and End of Sequence (EOS) tokens are used to start and stop graph generation.

IV. Graph Convolution Policy Network

Graph Convolution Policy Network (GCPN) proposed by You et al. in 2018 [6], a general graph convolutional network based model for goal directed graph generation through reinforcement learning. The model is trained to optimize domain-specific rewards and adversarial loss through policy gradient, and acts in an environment that incorporates domain-specific rules.

A. *Node Embeddings*

In order to perform link prediction in $G_t \cup C$, this model first computes the node embeddings of an input graph using Graph Convolutional Networks (GCN), a well-studied technique that



SOS
EOS
Sample + Edge-level Update
Node-level Update
Use start-of-sequence (SOS) token as the initial input.
*Usually a vector with all zeros / ones.
Let $x_{t+1} = y_t$
Use end-of-sequence (EOS) token as an extra RNN output.

Fig. 3: Graph Recurrent Neural Network.

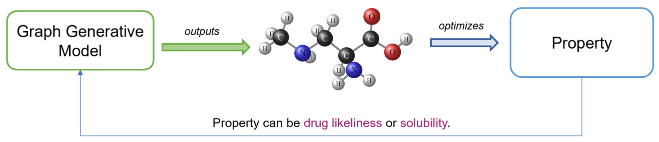


Fig. 4: Drug Discovery Process.

achieves very good performance in representation learning for molecules. GCPN uses the following variant that supports the incorporation of categorical edge types. The high-level idea is to perform message passing over each edge type for a total of L layers. At the l^{th} layer of the GCN, we aggregate all messages from different edge types to compute the next layer node embedding $H^{(l+1)} \in \mathbb{R}^{(n+c)\times k}$, where n, c are the sizes of G_t and C respectively, and k is the embedding dimension. In the same time, GCPN applies a L layer GCN to the extended graph $G_t \cup C$ to compute the final node embedding matrix $X = H^{(L)}$.

B. Action Prediction

The link prediction based action a_t at time step t is a concatenation of four components: selection of two nodes, prediction of edge type, and prediction of termination. Concretely, each component is sampled according to a predicted distribution governed by equation:

$$a_t = \text{CONCAT}(a_{\text{first}}, a_{\text{second}}, a_{\text{edge}}, a_{\text{stop}}) \tag{5}$$

The information from the first selected node a_{first} is incorporated in the selection of the second node by concatenating its embedding $Z_{a_{\text{first}}}$ with that of each node in $G_t \cup C$. Then an MLP maps the concatenated embedding to the probability distribution of each potential node to be selected as the second node. Note that when selecting two nodes to predict a link, the first node to select, a_{first}, should always belong to the currently generated graph G_t, whereas the second node to select, a_{second}, can be either from G_t (forming a cycle), or from C (adding a new substructure). To predict a link, another MLP takes $Z_{a_{\text{first}}}$ and $Z_{a_{\text{second}}}$ as inputs and maps to a categorical edge type. Finally, the termination probability is computed by firstly aggregating the node embeddings into a graph embedding using an aggregation function AGG, and then mapping the graph embedding to a scalar using an MLP.

C. Policy Gradient Training

Policy gradient based methods are widely adopted for optimizing policy networks. GCPN adopts Proximal Policy Optimization (PPO) as it is described in [9]. Pretraining a policy network with expert policies if they are available leads to better training stability and performance as shown in [10]. By looking at Figure 5, we can observe how cross entropy gradient and policy gradient work together for optimizing GCPN.

The application of GCPN can extend well beyond molecule generation. The algorithm can be applied to generate graphs in many contexts, such as electric circuits, social networks, and explore graphs that can optimize certain domain specific properties.

V. How to Evaluate Graph Generation

Evaluating the sample quality of generative models is a challenging task in general and evaluation requires a comparison between two sets of graphs (the generated graphs and the test sets). Whereas previous works relied on simple comparisons of average statistics between the two sets, a more accurate evaluation metrics that compare all moments of their empirical distributions.

These metrics are based on Maximum Mean Discrepancy (MMD) measures. The squared MMD between two sets of samples from distributions p and q can be derived as [11]:

$$\begin{aligned} MMD^2(p||q) = {} & \mathbb{E}_{x,y\sim p}[k(x,y)] + \mathbb{E}_{x,y\sim q}[k(x,y)] \\ & - 2\mathbb{E}_{x\sim p,y\sim q}[k(x,y)]. \end{aligned} \tag{6}$$

In the case of GCPN, the kernel function $k(x,y)$ will be the L_2 distance. To make it easier to compute, MMD using a set of graph statistics $\mathbb{M} = \{M_1, ..., M_k\}$, where each $M_i(G)$ is a univariate distribution over $\mathbb{R}$, such as the degree distribution or clustering coefficient distribution.

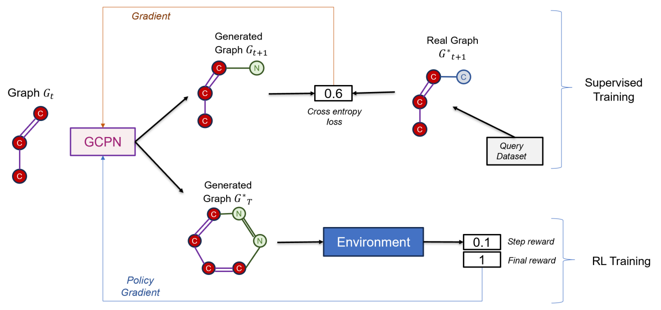


Fig. 5: Graph Convolution Policy Network.

REFERENCES

[1] Albert, R. and Barabasi, L. Statistical mechanics of complex networks. Reviews of Modern Physics, 74(1):47, 2002.

[2] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song. Syntax-directed variational autoencoder for structured data. arXiv preprint arXiv:1802.08786, 2018.

[3] Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In ICLR, 2014.

[4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In NIPS, 2014.

[5] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton and Jure Leskovec, GraphRNN: Generating Realistic Graphs with Deep Autoregressive Models, ICML 2018.

[6] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, Jure Leskovec, Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation, NeurIPS 2018.

[7] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen. Molecular denovo design through deep reinforcement learning. Journal of Cheminformatics, 9(1):48, Sep 2017.

[8] X. Yang, J. Zhang, K. Yoshizoe, K. Terayama, and K. Tsuda. ChemTS: An Efficient Python Library for de novo Molecular Generation. ArXiv e-prints, Sept. 2017.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. CoRR, abs/1707.06347, 2017.

[10] S. Levine and V. Koltun. Guided policy search. In International Conference on Machine Learning, 2013.

[11] Gretton, A., Borgwardt, K. M., Rasch, M. J., Sch¨ olkopf, B., and Smola, A. A kernel two-sample test. JMLR, 2012.