Conditional Graph Generation Using Neural Networks

2024-25 Data Challenge Report

Samuel Sarfati, Marouani Christopher, Mohammed Benyahia

15 January 2025

M2 - ALTEGRAD Course
DaSciM Group, LIX, École Polytechnique

Contents

1 Introduction

In this report, we present our approach to the Kaggle competition on generating graphs with specific properties. The goal of the project is to construct a graph from a textual description defining it's desired properties. The properties considered here are the number of nodes, edges, average degree, number of triangles, k-cores, communities, and the global clustering coefficient. The performance of the models is evaluated using the Mean Absolute Error (MAE) between the properties of the predicted graph and the ones of the textual description.

Currently, there is no algorithm that can construct itteratively a graph with this desired properties. Various algorithms have been developed for graphs generation such as **Erdős–Rényi Model (G(n, p))** that generates random graphs by specifying a number of nodes (n) and the probability (p) with which edges appear between them. Another approach, **Triadic Closure Algorithms**, ensures the formation of specific structures such as triangles or adjusts the clustering coefficient of the graph by adding edges in a way that promotes local connectivity.

Generative AI models such as GraphRNN have emerged as a breakthrough in graph generation tasks. The main ideas behin these models is to learn the distribution of the considered graphs and generate new graphs by sampling from the learned distribution.

The model that gave us the best MAE is an improvement of the baseline model that combined Variational Graph Autoencoder (VGAE) and a Latent Diffusion Model. The model was enhanced by conditioning the encoder and decoder on the properties of the graph and by regularizing the training with custom loss functions. To access coputation resources and facilitate collaboration, we used the Kaggle platform, which provides a Jupyter notebook environment with GPU support. The competition details are available here.

2 Dataset and Initial Observations

The training and validation sets consist of textual descriptions paired with their corresponding graphs, represented as lists of edges. These lists are processed into vectors: one containing the adjacency matrix, and another holding a feature vector that represents the eigenvector properties of the graph along with its seven key properties. The properties are extracted from the float values in the textual descriptions. Upon observing that the descriptions were highly similar, with properties appearing in the same order, we concluded that text encoding would not provide significant value.

- **Training Set**: 8,000 samples.
- **Validation Set**: 1,000 samples.
- **Test Set**: 1,000 textual descriptions for evaluation.

3 Architecture and Methodology

3.1 Model Architecture

Several approaches were explored as part of this challenge (see Section **??**), with the most effective one being based on the architecture proposed by [1], given as a baseline. Our modifications are described in detail in Section 3.2.

First, the VGAE compresses each input graph into a low-dimensional latent representation. The encoder employs message-passing layers, capturing both local interactions and more global

connectivity patterns. It then outputs parameters of a Gaussian distribution from which a latent vector is sampled. This probabilistic approach helps avoid overfitting to very specific graph structures, leading to a more robust and continuous latent space.

Next, the latent diffusion model operates on these compact representations. Rather than diffusing noise directly on high-dimensional adjacency matrices, the model adds noise to the latent vectors, creating a sequence of progressively perturbed samples. A denoising neural network then learns to remove the injected noise step by step. This denoising process is conditioned on an embedding of selected graph properties, which are encoded separately and fed into the denoiser. Once the final latent vector is sufficiently denoised, it is passed through the VGAE decoder to produce a complete adjacency matrix. This two-step approach effectively captures graph structure in a compact form while allowing new graphs to be generated under specified constraints.

3.2 Motivation and Intuition behind Enhancements

3.2.1 Loss Functions Impacts

We observed a significant improvement in performance when replacing the L1 norm with the Binary Cross-Entropy (BCE) loss for the reconstruction of the Variational Graph Autoencoder (VGAE). This adjustment appeared to regularize the training process, producing higher loss values that decreased more steadily. Additionally, since the data is binary, the BCE loss is inherently better suited for the task. Building on the insight that higher loss values can stabilize training, we ultimately chose to use the L1 norm with a summation parameter. This modification allowed us to achieve a Mean Absolute Error (MAE) score of 0.2.

3.2.2 Conditioning Improvements

Our new **decoder** introduces several improvements, including support for conditional input (**conditional**), which enhances reconstruction by leveraging the vectors that contain the properties of the graph. It also addresses missing values by imputing missing entries in the vector of properties with feature-wise means and normalizing inputs for robustness. After this, this vector is concatenated to the latent graph features. The architecture remains flexible, maintaining an MLP structure while dynamically adjusting the input size based on the conditional features. Overall, these changes improve reconstruction accuracy and stability, particularly when dealing with noisy or incomplete data.
We adjusted also the **encoder** such that we conditioned its inputs with the adjacency matrices of graphs and forwarding them to an MLP, and with the vector of properties pre-processed.
We applied the same pre-processing in the **denoiser** to the vector of properties before forwarding it to a light MLP and concatenating it with the latent features.

These changes make the model more expressive and suitable for real-world applications.

4 Model Tuning and Comparison

4.1 Evaluation Metrics

To access the performance of our model locally before submitting to Kaggle, we implemented a noramlized MAE metric. The MAE is calculated for each property and normalized by the range of the property values. The final score is the average of the normalized MAE across all properties.

4.2 Hyperparameter Optimization

Using this MAE noramlized metric we first did a grid search to optimize the hyperparameters of the baseline model. The hyperparameters include the number of epochs, the learning rate, the batch size, the number of layers, the number of hidden units, and the dropout rate.
Nevertheless we found that model producing the lower MAE was the one with the given parameters. Increasing the number of denoising epochs helps also to increase the performance.

4.3 Comparison

Table 1 summarizes the performance of various configurations of the model during the experimentation phase. The performance is measured in terms of MAE regarding the graph's 7 properties with the lowest value corresponding to the best configuration.

Table 1: Performance Comparison of Model Configurations

Description	**Details**	**Score**
Text encoder added	GloVe	0.883
Skip connections, $\beta = 0.01$	Decoder changes	0.82
Simple encoder + BCE	Linear layers instead of GIN	0.50
Baseline	L_1 sum	0.21
Decoder conditioning	+ Fine-tuning	0.15
Encoder conditioning	-	**0.14**

5 Approaches not very successful

5.1 Text Encoding :

In an attempt to enrich the model's understanding of graph properties, we explored encoding textual descriptions of the graphs. Instead of relying solely on numerical data, we employed pre-trained **GloVe** word embeddings to transform the textual descriptions into continuous vector representations. The TorchTextEncoder class was implemented to process the textual input, where each word in the description is represented by a GloVe embedding (100 dimensions). The embeddings of the words in a sentence were averaged to form a fixed-length sentence embedding. This approach allowed the model to leverage the semantic meaning of the textual descriptions alongside the numerical graph features. However, the MAE didn't improve a lot compared to the basic approach. An possible explanation would be that because in our particular case the text isn't relevant, because the numerical vector already captures the whole information.

5.2 Normalizing Flow

To enhance the latent space between the encoder and denoiser, we implemented a normalizing flow using coupling layers. The flow refines latent representations by applying a series of invertible transformations, modeled using learned scaling and translation functions. The input is split, and alternating parts undergo transformations through multiple feedforward layers with ReLU activations.
During training, the flow adjusts the latent space distribution, improving expressiveness and robustness. The loss function minimizes negative log-likelihood, helping the model learn a refined representation. This approach theoretically enhances the denoising process and better captures complex data distributions.However, in practice it wasn't really the case; the MAE didn't improve.

5.3 Hierarchical VAE

To improve performance, we tried a Hierarchical VAE, expecting its multi-level latent representations to better capture complex data. However, it performed badly compared to the standard VAE, with higher reconstruction and KLD losses. The added complexity likely introduced optimization challenges, so we retained the standard VAE for its balance of simplicity and effectiveness.

5.4 Loss Functions

For the reconstuction loss of the VGAE we also tried to regularized the training by adding a L2 norm to the L1 norm loss with a tuned gamma parameter. This didn't improve the performance of the model.

5.5 Data Augmentation

Considering that graph properties are invariant under node permutations, we implemented a data augmentation technique that randomly permutes the adjacency matrix of the graph mutliplying the dataset by 2, 4 and 8. Specifically, reindexing the nodes corresponds to permuting the rows and columns of the adjacency matrix. This technique was incorporated into the preprocessing function of the model. However, this approach did not lead to any improvement in the model's performance. We hypothesize that the model had already extracted all the learnable features from the given data. As a potential next step, we considered transitioning to a more complex architecture by adding additional layers or increasing the number of hidden units but decided to prioritize other avenues for improvement.

6 Conclusion and Future Work

In this report, we presented a framework for conditional graph generation, leveraging a combination of Variational Graph Autoencoders and Latent Diffusion Models. By integrating conditioning mechanisms and optimizing loss functions, we achieved significant improvements in the Mean Absolute Error (MAE), demonstrating the model's capability to generate graphs with specified properties.

While enhancements such as decoder conditioning and tailored preprocessing yielded robust performance, attempts at incorporating text encoding, hierarchical VAEs, and normalizing flows showed limited success, highlighting the need for task-specific solutions. Future work could explore more expressive latent spaces, domain-specific augmentations, or alternative conditioning strategies to further enhance graph generation fidelity.

7 Appendix

References

[1] I. Evdaimon et al. Neural graph generator: Feature-conditioned graph generation using latent diffusion models. 2024.

[2] X. Guo and L. Zhao. A systematic survey on deep generative models for graph generation. 2023.

[3] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

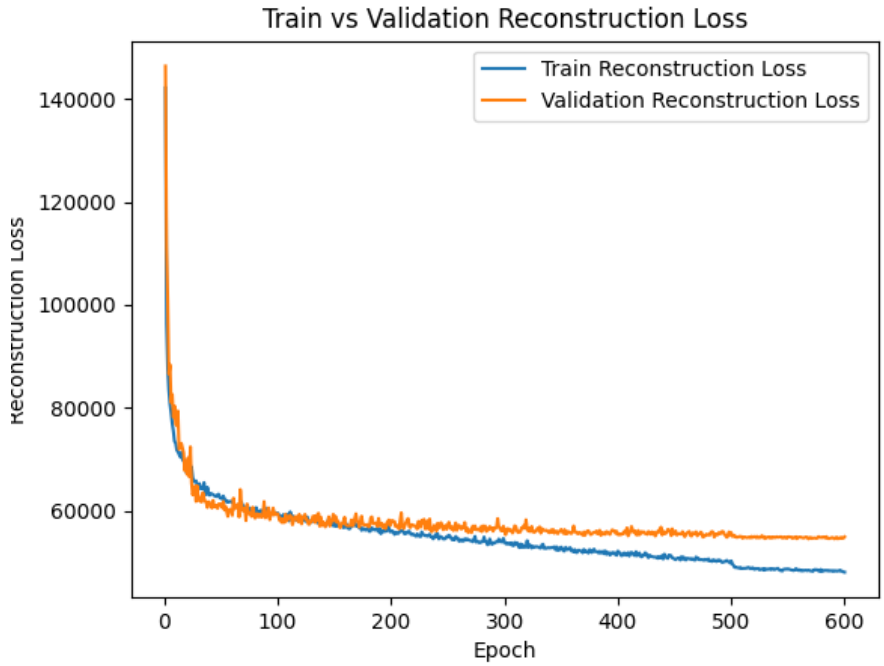

Figure 1: Reconstruction loss

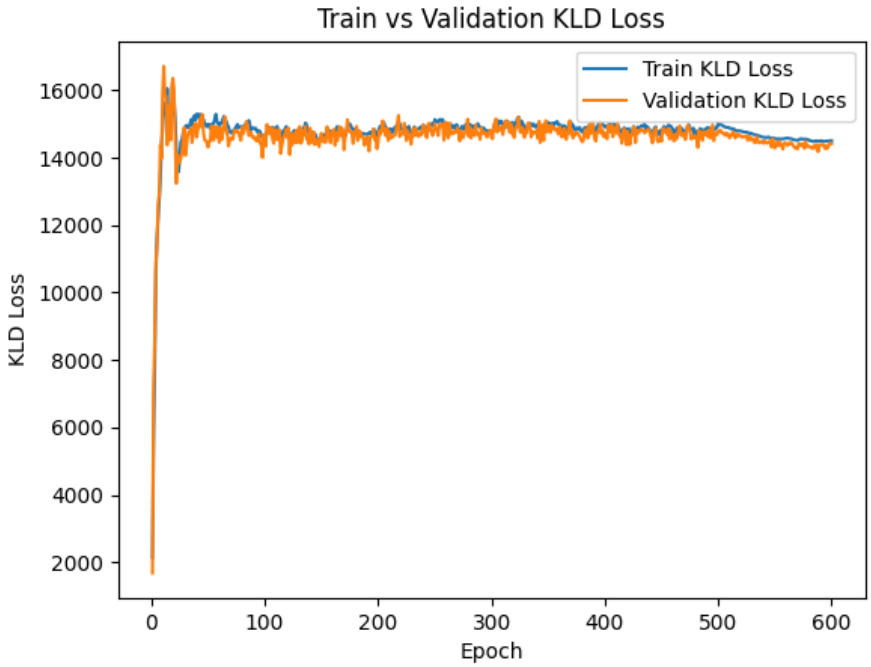

Figure 2: KLD loss.

[4] Y. Zhu et al. A survey on deep graph generation: Methods and applications. 2022.

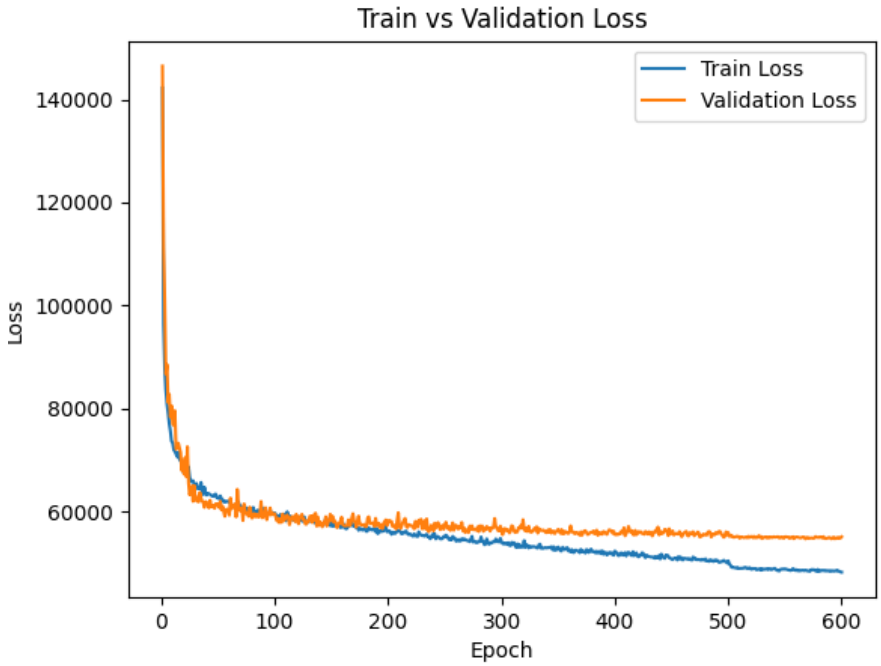

Figure 3: Autoencoder total loss

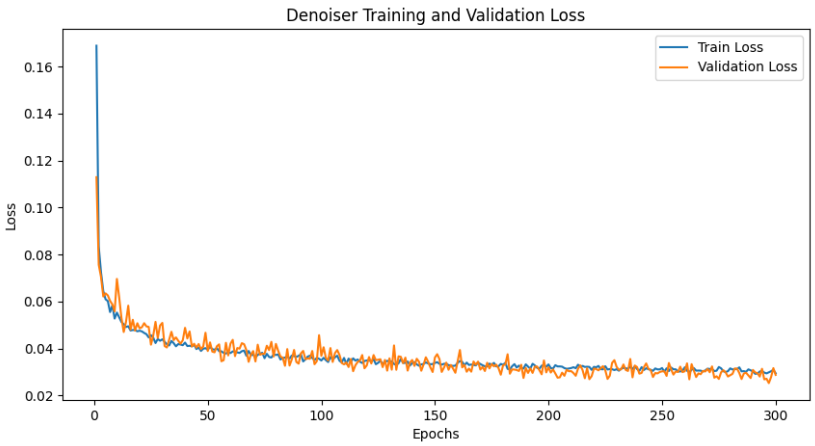

Figure 4: Denoiser loss plot