Neural Graph Generation Enhancement

Mohamed Amir Braham **Malek Bouzidi** **Philippe Gratias-Quiquandon**

1 Introduction

Graph generation is a critical area of research with a wide range of applications in diverse fields, including social network analysis, chemistry, biology, and computer science. For instance, in chemo-informatics, graph generative models are used to design novel molecular structures with desirable properties, such as high drug-likeness. The development of advanced machine learning models for graph generation has opened new possibilities for creating realistic and functional graphs, with recent progress emphasizing the use of deep generative techniques like variational autoencoders, generative adversarial networks, and diffusion models.

In this report, we address the challenge of generating graphs with specified properties based on textual descriptions. A baseline model described in Evdaimon et al. (2024) is already provided. Our task is to improve this model or modify certain components to achieve a better performance score. We explain here the multiple modifications we did, their interest and their results.

2 Enhancement Proposals

2.1 Loss for the VGAE

As a reminder, the adjacency matrix is made of 1 at the i, j (and the j, i as the matrix is symmetric) position if there is an edge between the node i and the node j otherwise it is 0. The default reconstruction loss implemented is a L1-loss which may be used for regression problems. It may be not optimal here as the values in the adjacency matrix are in fact binary (0 or 1) and not continuous in the range $[0; 1]$. Therefore, we can imagine that it would be more logical to change the loss from L1 to binary cross entropy ($L(x, \hat{x}) = \sum_n x_n \log(\hat{x}_n) + (1 - x_n) \log(1 - \hat{x}_n)$) to make it more efficient for classification of edges instead of regression.

It greatly improves the results, taking the MAE from 0.88 to 0.177. The binary cross-entropy loss was computed using a reduction set to 'sum' rather than the default 'mean' because we wanted the reconstruction term to scale with the total number of elements in the adjacency matrix. This choice ensures that the reconstruction loss reflects the overall magnitude of the graph data, which is particularly important when balancing it with the KL divergence term in the total loss function. This choice aligns with the need for a scale-compatible contribution of the reconstruction loss and the KL divergence in the total loss. Something else about the loss is this β factor, we did a grid-search and found the best validation score with $\beta = 0.05$.

2.2 Features extraction

By looking at some textual prompts and the file "extract_feats.py", we see that we just extract the integers and the floats in the order they appear without further processing. However, there are details that we do not extract, for example, the sentence "`Within the graph, there are 55 triangles, forming closed loops of nodes`" has two pieces of information: the number of triangles and their form (closed loops of nodes). When we look through all the data, it is either the triangles are forming closed loops of nodes or nothing.

We added a system that add another binary feature that is 0 if there is no closed loops of nodes, and a positive value otherwise. Fortunately, the order of the numbers and their features is always the same

Improvement	MAE	Details
Default setup (L1 loss)	0.88	No adjustments made.
Switch to Binary Cross-Entropy (BCE) and grid-search on β	0.177	Reconstruction loss changed to BCE, computed with reduction='sum'.
Feature extraction improvement (closed loops)	0.172	Added binary feature for closed loops, refined extraction with SpaCy-based NLP checks.
Decoder Modifications and DenoiseNN Simplifications	0.171	SSigmoid instead of Gumbel-Softmax, enforcing symmetry in the adjacency A refined DenoiseNN that concatenates cond and time embedding in a simpler way, A new Decoder class that outputs adjacency matrices via a Sigmoid and upper-triangle filling since we know that the adj matrix is symetrical
Simple VAE	0.158	Simple VAE which takes the features vector (dimension 8 for our case) as the input and generates a graph
Increased number of epochs	0.13	Changed the number of epochs for both the autoencoder and the denoiser to 6000.
Using BERT to generate vector embeddings from text	0.164	The number of features is now 8+768. We have used bert-base-uncased without doing any finetunning
Cross-attention mechanism	1.1	Adding a Cross-Attention mechanism to the denoiser instead of concatenation for conditioning.
Increased complexity of encoder and reducing the number of epochs	**0.128**	Adding a layer, increasing the size of the hidden dimensions for the encoder and trained during 4000 epochs both the autoencoder and the denoiser
Generate the same graph several times and computing the absolute error	**0.076**	Generating the same graph multiple times and choosing the best, by computing the absolute error between the 7 ground-truth features and the features of the graph

Table 1: Summary of Improvements

through all the sentences, so that is why the simple regex system worked very well. However to make the extraction more robust, we added some NLP processing (using SpaCy) that looks for the words that precede the digits to make sure the features vector is always in a good order. The results were a bit better, the MAE went from 0.177 to 0.172

2.3 A simpler architecture

In addition to the improvements mentioned above, we also experimented with a simpler architecture: a standard variational autoencoder (VAE) designed to directly process the feature vector extracted for each graph. This approach simplifies the entire process by focusing on encoding the graph's 8-dimensional feature vector into a latent space and subsequently decoding it to reconstruct the graph structure.

The encoder used in this VAE consists of a simple feedforward neural network with two fully connected layers: The input 8-dimensional feature vector is first passed through a fully connected layer with a linear transformation, followed by a BatchNorm1D operation and a ReLU activation. This is followed by another linear transformation that maps the features into a low-dimensional latent space. As in standard VAE implementations, the latent space is regularized using the Kullback-Leibler

divergence (KLD) loss to ensure that the latent variables follow a Gaussian distribution. The decoder reconstructs the graph from the latent vector in a manner similar to the VGAE's decoder. The loss is exactly the same as the baseline model as we have access to the ground truth graphs but we do not use them as input in the auto-encoder.

The key difference lies in the significantly reduced number of parameters compared to the baseline model, which included a complex latent diffusion process. By omitting the diffusion component and using a simpler feedforward structure, we greatly reduced the computational complexity while still achieving good results. By doing so, the autoencoder continues to reconstruct the graph structure in latent space while simultaneously encouraging robust and invariant latent representations.

Models	**Number of parameters**
Baseline Model	1,697,906
VAE	705,426
With Augmented Encoder	1,785,842
With Cross-Attention	1,852,530

Table 2: Summary of the number of Parameters

Despite the simplicity of this architecture, it yielded competitive performance with a MAE of 0.158. This suggests that the simpler VAE model, while lacking the sophisticated graph generation capabilities of the baseline, can serve as a lightweight alternative with far fewer parameters. This makes it a viable option for applications where computational efficiency is a critical constraint.

2.4 Cross-Attention for Conditioning in the Denoising Model

To improve the conditioning mechanism in the denoising model, we replaced the simple concatenation of the conditioning features with a cross-attention mechanism Gheini et al. (2021). Indeed, the condition vector has 8 dimensions and we can imagine that if we concatenate this small vector to the noisy latent vector, it will be ignored as $n_{\text{cond}} \ll d_{\text{latent}}$. So, in the baseline code, we project it to a dimension named d_{cond} which takes the dimension from 8 to 128 to give the features more importance during the denoising.

We thought that we could improve this concatenation by using cross-attention. What it basically does is running a multi-attention head over all the latent vector with the features. Therefore, we implemented it using the trainable keys, queries and values mechanisms and trained only the denoiser for 4000 epochs. However, the results were not as competitive as the others. It could be explained as we decided to keep the projection of the feature vector to $d_{\text{cond}} = 128$ which was probably a mistake. Unfortunately, we did not keep this mechanism as the results were really terrible but there is still room for improvement.

2.5 Multiple graph generations and error computing

We obtained our best score with 4000 epochs and the augmented encoder explained in Table 1. The final score is calculated based on 15 features but we already have 7 in the prompt. From this came the idea of computing the Mean Absolute Error (MAE) on the **7 features** to have an estimation on these features. This helped us to know if a run was good or not. Then we used this to make multiple generations of the same graph, compute the absolute error for each graph and keep the one which has the smallest one. This greatly decreased the error, giving our best score of 0.076.

2.6 Contrastive Graph Augmentation

In addition to the architectural and loss-function enhancements described above, we explored a *contrastive learning* strategy to further improve the quality of latent representations learned by the Variational Graph Autoencoder (VGAE). Specifically, we introduced a **graph augmentation** step and a **contrastive objective** based on the NT-Xent (Normalized Temperature-scaled Cross Entropy) Sohn (2016) loss. This approach is inspired by the contrastive frameworks widely used in vision and

graph neural network research, where multiple augmented "views" of the same input are encouraged to be similar in the learned embedding space.

2.6.1 Graph Augmentation

Given an input graph G, we create two random augmentations ("views") G_1 and G_2 via a stochastic process that applies:

- **Node Dropping**: Each node is dropped with a probability p_{node}, along with its incident edges, effectively sampling a random subgraph.
- **Edge Dropping**: Independently, each edge is dropped with a probability p_{edge}, reflecting additional uncertainty or noise in connections.
- **Feature Masking**: A fraction of node features are randomly zeroed out, further diversifying the two augmented views.

By applying these perturbations, G_1 and G_2 retain the same high-level semantics as G, but differ in specific nodes, edges, and masked features. This randomness drives the contrastive signal, making the encoder invariant to small perturbations in graph structure and features.

2.7 Contrastive Objective via NT-Xent

Let $\mathbf{z}_1$ and $\mathbf{z}_2$ be the latent embeddings of the augmented graphs G_1 and G_2, respectively, after passing through the encoder of the VGAE. We introduce an additional projector network $f(\cdot)$ (an MLP) that maps embeddings into a space where the contrastive loss is applied. Denoting $\mathbf{z}'_1 = f(\mathbf{z}_1)$ and $\mathbf{z}'_2 = f(\mathbf{z}_2)$, we compute the NT-Xent loss as:

$$\ell_i = -\log \frac{\exp\left(\mathrm{sim}(\mathbf{z}'_1, \mathbf{z}'_2)/\tau\right)}{\sum_{k=1}^{N} \exp\left(\mathrm{sim}(\mathbf{z}'_1, \mathbf{z}'_k)/\tau\right)},$$

where $\mathrm{sim}(\cdot,\cdot)$ is the cosine similarity, τ is a temperature hyperparameter, and the denominator sums over N samples in the batch (including both positive and negative pairs). Averaging this loss over the batch promotes $\mathbf{z}'_1$ and $\mathbf{z}'_2$ of the same graph to be close, while different graphs'embeddings remain relatively separated.

2.7.1 Combined Training Objective

During training, we combine the standard VGAE reconstruction loss (either L1 or BCE on the adjacency matrix) and the KL-divergence term with the above contrastive loss. Let $\mathcal{L}_{\text{VGAE}}$ denote the baseline VGAE objective (reconstruction + KL), and let $\mathcal{L}_{\text{contrast}}$ denote the NT-Xent-based contrastive loss. We use a balancing hyperparameter λ:

$$\mathcal{L} = \mathcal{L}_{\text{VGAE}} + \lambda\, \mathcal{L}_{\text{contrast}}.$$

However, the results were not better and not very different. We did not go further in this topic to improve the model.

2.8 DenoiseNN Simplifications

To improve our baseline model, we decided to make some changes to the denoising neural network (DenoiseNN). Our goal was to simplify this components, reduce the number of parameters, and enhance the overall performance of the model.

The DenoiseNN is crucial for the latent diffusion process as it refines noisy latent vectors into clean representations. The original network was quite complex, which made training more challenging and increased the risk of overfitting. To streamline this, we implemented the following changes:

- **Simpler Concatenation**: Instead of a complicated method for combining conditional information and time embeddings, we opted for a straightforward concatenation before passing them through the network. This reduced the number of parameters and made the training process faster.

- **Reduced Architecture**: We cut down the number of layers and neurons in the DenoiseNN. By using a smaller multi-layer perceptron (MLP), we maintained the network's ability to denoise effectively while making it more efficient.
- **Efficient Activation Functions**: Switching to ReLU activations helped the network converge faster and improved its ability to capture non-linear relationships without adding significant computational load.

2.8.1 Impact on Performance

Simplifying the DenoiseNN reduced the number of parameters. Despite having fewer parameters, the Mean Absolute Error (MAE) improved slightly from 0.177 to 0.171. This indicates that our streamlined approach retained the model's ability to reconstruct graphs effectively while being more computationally efficient.

2.9 Using BERT to Generate Vector Embeddings from Text

To make our model better at understanding and utilizing textual descriptions for graph generation, we integrated BERT (Bidirectional Encoder Representations from Transformers) Devlin et al. (2019) to create high-dimensional vector embeddings from text inputs. BERT's strong language understanding capabilities helped us capture more nuanced information from the text.

2.9.1 Integrating BERT Embeddings

In the baseline model, we were simply extracting numbers from the text without really understanding the context. To address this, we decided to use the `bert-base-uncased` model to generate embeddings that capture the meaning and context of the text. Here's how we did it:

1. **Preprocessing Text**: We cleaned the textual prompts by removing any irrelevant characters and ensuring consistent formatting.
2. **Generating Embeddings**: The cleaned text was fed into BERT, which produced a 768-dimensional embedding vector for each input. This vector encapsulates the semantic information of the text.
3. **Augmenting Feature Vectors**: We combined the original 8-dimensional feature vector with the 768-dimensional BERT embedding, resulting in a 776-dimensional input vector for the encoder.

2.9.2 Adjusting the Model

With the feature vector size increasing from 8 to 776 dimensions, we had to adjust our encoder accordingly. We modified the encoder to handle the larger input size by expanding the initial fully connected layers. This ensures that the encoder can effectively process the richer feature vector. Given the high dimensionality, we added regularization techniques like dropout and batch normalization. These help prevent over-fitting and ensure that the model learns robust representations.

3 Conclusion

In this work, we explored various improvements and potential pitfalls in our approach, acknowledging that some outcomes may be influenced by implementation challenges. Despite these hurdles, the underlying idea remains promising and worth further exploration. Among the tested strategies, the most significant advancement was the adoption of binary cross-entropy loss for the adjacency matrix, which yielded substantial improvements in model performance. Beyond this, the remaining adjustments primarily involved fine-tuning parameters and refining architectural details.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL] `https://arxiv.org/abs/1810.04805`

Iakovos Evdaimon, Giannis Nikolentzos, Christos Xypolopoulos, Ahmed Kammoun, Michail Chatzianastasis, Hadi Abdine, and Michalis Vazirgiannis. 2024. Neural Graph Generator: Feature-Conditioned Graph Generation using Latent Diffusion Models. arXiv:2403.01535 [cs.LG] https://arxiv.org/abs/2403.01535

Mozhdeh Gheini, Xiang Ren, and Jonathan May. 2021. Cross-Attention is All You Need: Adapting Pretrained Transformers for Machine Translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 1754–1765. https://doi.org/10.18653/v1/2021.emnlp-main.132

Kihyuk Sohn. 2016. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf