Mekki DAMAK
mekki.damak@polytechnique.edu
Mariem doggaz
mariem.doggaz@ensta-paris.fr
Omar maalej
omar.maalej@ensta-paris.fr

Generating Graphs with Specified Properties
ALTEGRAD 2024

01/15/25

1 Overview

This report details the work undertaken for the Kaggle competition titled ”Generating Graphs with Specified Properties.” The primary goal of the project is to utilize advanced machine learning and artificial intelligence techniques to generate graphs conditioned on textual descriptions. The challenge of graph generation holds significant relevance across diverse fields, including chemo-informatics, where the task involves generating molecular graphs with high drug-likeness or other desirable properties. This project involves training a latent diffusion model to translate textual descriptions into corresponding graph structures.

2 Introduction

Graph generation is an emerging domain in machine learning with applications in fields such as social network analysis, cheminformatics, and recommendation systems. The aim of this competition is to design and implement a model capable of generating graphs based on textual prompts. A successful implementation must balance two competing objectives: creating novel graphs and ensuring these graphs align with their described structural and property requirements.

The dataset provided by the competition is organized into training, validation, and test sets. Each sample in the training and validation datasets consists of a textual description and a graph file. The textual description encapsulates information about the graph's structure and properties, while the graph file is provided in both edgelist and GraphML formats. The test dataset, however, includes only descriptions, and the task is to predict the corresponding graph properties for these samples.

The performance evaluation of generated graphs is conducted using Mean Absolute Error (MAE) between the target graph properties and the properties of the generated graphs. A lower MAE indicates closer adherence to the target properties.

3 Dataset Description

The dataset comprises three components:

- **Training Set:** Contains 8,000 samples with graph files and corresponding textual descriptions. The graph files are stored in edgelist and GraphML formats.
- **Validation Set:** Includes 1,000 samples in the same format as the training set.
- **Test Set:** Contains 1,000 textual descriptions in test.txt. The descriptions are paired with the predicted graph outputs for evaluation.

Each description highlights key aspects of the graph, including the number of nodes, edges, degree distributions, clustering coefficients, and other structural properties. This information serves as input for the model, which generates corresponding graph structures.

4 Methodology

The methodology adopted in this project combines a Variational Graph Autoencoder (VGAE) and a diffusion-based model to generate graphs aligned with the provided textual descriptions. The primary goal is to simultaneously capture and leverage the structural information of graphs and the semantic constraints from textual data.

4.1 Data Preprocessing

Data preprocessing is a critical step to ensure that the graph and textual data are structured and enriched before being fed into the model.

4.1.1 Graph Processing

The graphs in the training and validation sets are processed as follows:

- **Graph Loading:** Graphs are imported using the `networkx` library and converted into adjacency matrices. These matrices represent node connectivity in a binary format.
- **Standardization:** A Breadth-First Search (BFS) is applied to standardize the structure of graphs, ensuring consistency across samples.
- **Spectral Feature Extraction:** Eigenvalues and eigenvectors of the graph Laplacian matrix are computed to provide a compact and structured representation of the graph's topology.
- **Graph Padding:** Graphs with fewer nodes than the maximum allowable size (*n_max_nodes*) are padded with zeros to standardize input sizes.

4.1.2 Textual Description Processing

Textual descriptions provide essential semantic information and are processed as follows:

- **Text Encoding:** A pre-trained model (`jinaai/jina-embeddings-v3`) is used to generate **text embeddings**, which are dense vector representations of the descriptions. These embeddings capture the semantic meaning of the text and establish a connection between textual and structural data.
- **Normalization and Structuring:** The textual embeddings are aligned with graph features to ensure a seamless integration of semantic and structural data.

4.1.3 Data Structuring

The processed graph and textual data are structured into **torch_geometric.data.Data** objects, containing:

- The graph adjacency matrix.
- Spectral and structural features of the graph.
- Text embeddings representing the descriptions.

This structuring facilitates seamless integration into the learning models.

4.2 Model Training

Two primary models are trained to achieve the project objectives:

4.2.1 Variational Graph Autoencoder (VGAE)

The VGAE encodes and reconstructs graph structures. The key steps are:

- **Encoder:** The custom **GraphEncoder_1**, based on Graph Convolutional Network (GCN) layers, encodes graphs into a compact latent space. Layer normalization and residual connections enhance stability and performance.
- **Reparameterization:** The latent distribution is constrained to follow a Gaussian distribution using the Kullback-Leibler Divergence (KLD).
- **Decoder:** Graphs are reconstructed from latent representations using a Multi-Layer Perceptron (MLP)-based decoder.
- **Loss Functions:**
  - **Reconstruction Loss (L1 Loss):** Measures the difference between the reconstructed and original adjacency matrices.
  - **KL Divergence Loss:** Regularizes the latent space.

4.2.2 Diffusion Model

The diffusion model refines latent representations to ensure generated graphs align with textual descriptions:

- **Architecture:** The **DenoiseNN** model integrates text embeddings at each layer, enabling the diffusion process to incorporate textual constraints.
- **Forward Diffusion:** Noise is progressively added to latent representations following a linear beta schedule (*linear beta schedule*).
- **Reverse Diffusion:** The model learns to denoise and reconstruct latent graphs, aligning them with the descriptions.
- **Huber Loss:** Used for noise prediction, combining the robustness of L1 Loss and the stability of L2 Loss.

4.3 Graph Generation

Once trained, the models generate graphs as follows:

- **Latent Sampling:** Latent representations are generated using the diffusion model.
- **Decoding:** Latent representations are converted into adjacency matrices by the VGAE decoder.
- **Post-processing:** The adjacency matrices are transformed into NetworkX graphs for analysis and storage.
- **Optimal Selection:** Among multiple graphs generated for each description, the one minimizing the Mean Absolute Error (MAE) is selected.

5 Model Architectures

5.1 Variational Graph Autoencoder (VGAE)

The Variational Graph Autoencoder (VGAE) is a key component of the architecture, tasked with encoding graph structures into a latent representation and decoding them back into adjacency matrices. Its structure comprises the following elements:

- **Encoder:** The encoder leverages the **GraphEncoder_1** module, which is built using five Graph Convolutional Network (GCN) layers. These layers perform localized graph convolutions to aggregate neighborhood information.
- **Latent Space Representation:** The encoder outputs two parameters, μ and $\log \sigma^2$, which define a Gaussian distribution. The reparameterization trick is applied to sample from this distribution during training, ensuring differentiability.
- **Decoder:** The decoder reconstructs the graph's adjacency matrix from the latent representation. It utilizes a Multi-Layer Perceptron (MLP) with multiple layers and applies the Gumbel-Softmax function to generate a binary adjacency matrix.

The VGAE is trained using a combination of two losses:

- **Reconstruction Loss:** Measures the difference between the reconstructed adjacency matrix and the original matrix using L1 loss.
- **KL Divergence Loss:** Regularizes the latent space to follow a standard normal distribution.

5.2 Updated Encoder: GraphEncoder_1

The encoder module, **GraphEncoder_1**, replaces the original GIN-based encoder, introducing architectural improvements specifically tailored to enhance graph representation learning for the task of generating graphs with specified properties. Its design leverages **Graph Convolutional Networks (GCNs)** to aggregate information from multi-hop neighborhoods, capturing both local and global graph features.

Architecture **GraphEncoder_1** incorporates the following key elements:

- **GCN Layers:** Five GCN layers propagate and process node-level features across the graph. Unlike the GIN encoder, which focuses on local node neighborhoods, GCN layers integrate multi-hop connectivity patterns, making them more effective at learning complex structural relationships.
- **Residual Connections:** By adding residual connections between layers, **GraphEncoder_1** improves gradient flow and facilitates the training of deeper networks, addressing the vanishing gradient problem.
- **Global Mean Pooling:** The encoder aggregates node embeddings into a cohesive graph-level representation through global mean pooling. This operation ensures that both local and global graph properties are captured efficiently.
- **Fully Connected Layers:** Two fully connected layers refine the graph-level representation, providing the flexibility to adapt the learned features to downstream tasks.
- **Layer Normalization and Dropout:** To stabilize training and prevent overfitting, layer normalization is applied after each layer, and dropout is used as a regularization technique.

Advantages Over GIN Encoder The following aspects make **GraphEncoder_1** better suited for this problem compared to the original GIN encoder:

- **Global Information Integration:** GCNs aggregate multi-hop neighborhood information, capturing complex structural dependencies. In contrast, GIN focuses on localized node features, which may miss global patterns crucial for property alignment.
- **Robustness to Graph Variability:** The global pooling mechanism in **GraphEncoder_1** ensures that diverse graph structures are effectively represented, reducing sensitivity to variations in graph topology.
- **Improved Training Stability:** Residual connections in **GraphEncoder_1** mitigate issues with vanishing gradients, ensuring smooth optimization and better convergence.
- **Feature Enrichment:** The additional GCN layers enable the model to learn deeper structural patterns, while layer normalization and dropout prevent overfitting, especially important given the complex and diverse nature of the dataset.

Impact on Performance By integrating these architectural advancements, **GraphEncoder_1** demonstrated a significant **15% reduction in the Mean Absolute Error (MAE)** compared to the GIN encoder. This improvement highlights its superior ability to encode graph structures and properties, making it a more effective solution for the task of generating property-aligned graphs.

Relevance to the Problem The task of generating graphs based on textual descriptions requires a model that can understand both global structural patterns and fine-grained details. The **GIN encoder**, while effective at capturing local features, falls short in modeling global dependencies. **GraphEncoder_1**'s GCN-based design bridges this gap by integrating global and local features, ensuring a closer alignment between the generated graphs and their textual descriptions.

5.3 Improvements to the Denoising Model

The denoising model, **DenoiseNN**, has been significantly enhanced to improve its performance and alignment with textual constraints. A cornerstone of these improvements is the integration of the pre-trained `jinaai/jina-embeddings-v3` model for generating text embeddings.
Among the vast range of pre-trained language models, `jinaai/jina-embeddings-v3` was selected for its efficiency, domain adaptability, and capability to generate high-quality semantic embeddings. Its lightweight nature ensures fast processing, while its ability to capture nuanced semantic relationships makes it ideal for aligning textual descriptions with graph properties. This choice simplifies the preprocessing pipeline, replacing traditional approaches with dense, expressive embeddings that enhance the model's learning process.

Key Modifications to DenoiseNN:

- **Text Embedding Integration:** Embeddings generated by `jinaai/jina-embeddings-v3` are directly incorporated into the noise prediction process at each layer. This enables the model to better understand and encode the semantic constraints of textual descriptions, ensuring generated graphs adhere to the described properties.

- **Batch Normalization (BN):** Batch normalization is applied to each layer to stabilize training by normalizing feature distributions. This reduces sensitivity to initialization and accelerates convergence.
- **Huber Loss:** The adoption of Huber loss for noise prediction balances the robustness of L1 loss and the stability of L2 loss, making it particularly effective in handling noisy data.
- **Sinusoidal Position Embeddings:** Time-step information is encoded using sinusoidal embeddings, which are fed into the model alongside graph and text data. This ensures temporal coherence during the diffusion process.

Impact of Updates: These updates collectively enhance the denoising model's ability to align the generated graphs with their textual descriptions. By integrating semantic-rich text embeddings, robust loss functions, and stable training techniques, the model achieves a significant improvement in the quality of the generated graphs, as evidenced by a substantial reduction in the Mean Absolute Error (MAE).

6 Results and Evaluation

The models were evaluated on the validation set using the Mean Absolute Error (MAE) between the properties of the generated graphs and the target properties. The evaluation results demonstrate the effectiveness of the proposed architecture:

- **Impact of Text Embedding Integration:** Incorporating text embeddings into the denoising model led to a dramatic reduction in MAE, dropping from 0.8 to 0.3. This demonstrates the critical role of semantic constraints in aligning generated graphs with textual descriptions.
- **Reduction in MAE with GraphEncoder_1:** By replacing the GIN-based encoder with **GraphEncoder_1**, the MAE improved significantly, decreasing from 0.3 to 0.2. This highlights the encoder's superior capability in capturing graph structural features and its suitability for this task.
- **Overall Model Performance:** Through additional architectural enhancements and hyperparameter tuning, the combined system of VGAE and the denoising model achieved a further reduction in MAE to 0.17, underscoring the effectiveness of the proposed improvements.

7 Conclusion

This project successfully addressed the challenge of generating graphs conditioned on textual descriptions. By integrating advanced components such as pre-trained Jina embeddings, a robust graph encoder (GraphEncoder_1), and an enhanced denoising model, the proposed architecture significantly outperformed baseline approaches. The results underline the potential of combining semantic understanding with structural learning for graph generation tasks.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[3] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[4] Ritvik Teelucksingh. Papers explained: Jina embeddings v3, 2023. Accessed: 2025-01-14.