Generating Graphs with Specified Properties
ALTEGRAD 2025

BESBES Mohamed Amine
MKAOUAR Yassine
JLASSI Eya

Data Science Master

2025

Problem Statement:

Overview

In our problem setting, we are provided with a dataset containing graphs and their corresponding textual descriptions.

1. The objective is to develop a model that, given a textual query describing a graph, generates a graph that best aligns with the description.

Introduction:

- The **training** dataset includes **8,000** graph-description pairs, with **1,000** pairs reserved for validation and another **1,000** for testing.
- For evaluation, a vector representing the seven required properties of each generated graph is constructed and compared to the corresponding properties specified in the textual description
- The goal is to minimize the **Mean Absolute Error (MAE)**, ensuring the generated graphs closely align with the specified properties.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Where:

- n is the number of properties being evaluated.
- y_i is the actual property value mentionned in the textual description.
- $\hat{y}_i$ is the corresponding property value of the generated graph.

Introduction:

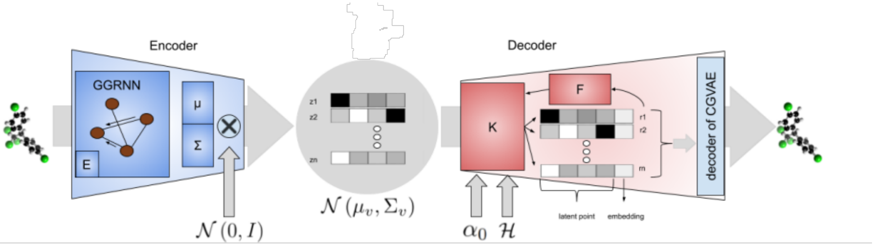
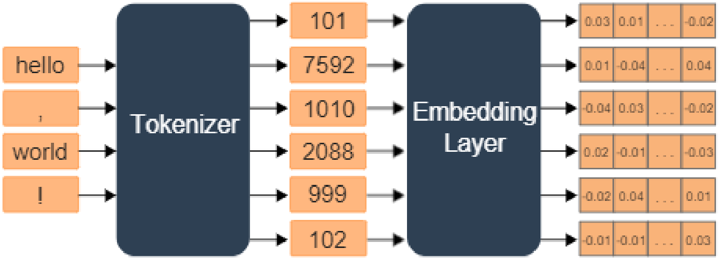


Figure: Conditional Constrained Graph Variational Autoencoder model structure [1]

Feature Processing and Embedding Techniques

Semantic Embedding with BERT

- **BERT**: A pre-trained language model developed by Google that leverages the Transformer architecture to produce context-aware representations of words, phrases, or sentences from large-scale text data.



Feature Processing and Embedding Techniques

Semantic Embedding with BERT

- We used BERT to encode graph descriptions into 768-dimensional semantic embeddings. This approach replaced the traditional input of only 7 numerical features (e.g., number of nodes, edges, average degree) with richer, context-aware embeddings.
- By encoding the graph descriptions with BERT and utilizing these embeddings in the training pipeline, the model's performance, measured by MAE, improved significantly from **0.88** to **0.35**.

Feature Processing and Embedding Techniques

Transition to Gemini API for Semantic Embeddings

- While BERT significantly improved the representation of graph descriptions, it has limitations as a relatively older model.
- This prompted us to explore more advanced and modern embedding techniques.

Feature Processing and Embedding Techniques

Adopting Gemini API's `text-embedding-004`

- The Gemini API's `text-embedding-004` model provides state-of-the-art continuous vector representations for text.
- Its embeddings are optimized for capturing fine-grained semantic differences, making it well-suited for our use case.
- By integrating Gemini API embeddings, we achieved richer and more consistent vector representations of graph descriptions.
- The model's performance, measured by MAE, improved significantly from **0.35** to **0.24**.

Feature Processing and Embedding Techniques

Enhancing Embeddings with Numerical Features

- However, the embeddings alone may overlook critical numerical details, such as the number of nodes, edges, and clustering coefficients, which are vital for distinguishing graph structures.
- To address this, we propose augmenting the semantic embeddings with normalized numerical features to create a comprehensive representation.

Feature Processing and Embedding Techniques

Enhancing Embeddings with Numerical Features

- However, the embeddings alone may overlook critical numerical details, such as the number of nodes, edges, and clustering coefficients, which are vital for distinguishing graph structures.
- To address this, we propose augmenting the semantic embeddings with normalized numerical features to create a comprehensive representation.

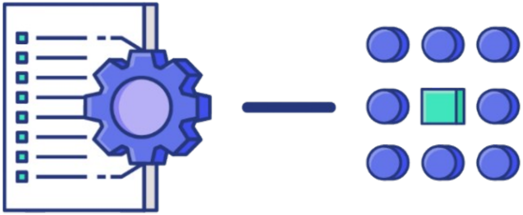
Proposed Approach

- Extract numerical features directly from the graph descriptions using regular expressions.
- Normalize these features to ensure consistency and scale them to align with the dimensions of the semantic embeddings.
- Concatenate the normalized numerical features with the `text-embedding-004` vectors.

Feature Processing and Embedding Techniques

Results

- The Mean Absolute Error (MAE) was significantly reduced from **0.24** to **0.14**.
- This improvement highlights the importance of leveraging both semantic and numerical information for downstream tasks.



Variational Auto-encoder

Variational Auto-encoder

In our model, the Variational Autoencoder (VAE) acts as the backbone of the generative process. It is designed to encode graph structures into a compact latent space representation and decode them to reconstruct or generate new graphs. The VAE architecture is composed of two main components: an encoder and a decoder.

Variational Auto-encoder

The encoder

- Our solution served as the starting point: we worked with the encoder, implemented as a **Graph Isomorphism Network** (GIN), to extract structural features from the input graph.
- These features were summarized into two key outputs: the mean (μ) and the log variance ($\log \sigma^2$), which parameterize the latent distribution of the graph.
- Using these outputs, we applied the reparameterization trick to sample a latent vector (z) while maintaining differentiability during training.

Variational Auto-encoder

GIN

The encoder was designed using a multi-layer GIN, which aggregates node features by considering their neighbors and applying transformations through fully connected layers, batch normalization, and LeakyReLU activations.
By stacking multiple GIN layers, the encoder effectively captured complex structural patterns within the graph, producing a compact latent representation.

Variational Auto-encoder

The decoder

The decoder utilizes the latent vector to reconstruct the graph's adjacency matrix. In addition, our model integrates text embeddings into the latent space, enabling the decoder to condition graph generation on both the graph's encoded structure and its textual description, ensuring alignment with the provided semantic context.

Variational Auto-encoder

The decoder

The decoder utilizes the latent vector to reconstruct the graph's adjacency matrix. In addition, our model integrates text embeddings into the latent space, enabling the decoder to condition graph generation on both the graph's encoded structure and its textual description, ensuring alignment with the provided semantic context.

MLP

The decoder consists of fully connected layers, Batch Normalization, and ReLU activations. The final layer reshapes the output into an adjacency matrix using a Gumbel-softmax function, preserving the graph structure. Furthermore, we enhanced the decoder by incorporating dropout layers to reduce overfitting and expanding the latent space dimensions to improve the representation capacity.

Variational Auto-encoder

But we didn't leave the model as it was, we experimented with a number of parameter changes . In the end, the most efficient configuration was determined as follows:

- **Increasing the dropout rate** to enhance generalization (from 0.0 to 0.3).
- **Expanding the hidden dimensions** of the encoder (from 64 to 256), decoder (from 256 to 512), and latent space dimension (from 32 to 64) for better representation capacity.
- **Increasing the number of hidden layers** in both the encoder (from 2 to 5) and decoder (from 3 to 6) to capture more complex patterns.

These adjustments were crucial in improving the model's performance and robustness.

Graph Generation

Challenges

Despite training the model, several challenges can arise during graph generation:

- **Mismatch with Target Properties:** Some generated graphs fail to align with the specified properties due to inaccuracies in the model's predictions.
- **Incomplete or Invalid Properties:** Missing or invalid values for certain graph properties can render the graph unsuitable.
- **Scale Disparities:** Different properties, such as node count and clustering coefficient, operate on vastly different scales, leading to biased evaluation.

Graph Generation

Solution

To address these issues, we introduced a quality control mechanism that:

- Generates n_tries number of graphs.
- Returns the graph that best matches the textual description.

Question:

- How can we identify the most accurate graph that aligns with the specified properties?

Graph Generation

Steps

1. **Property Extraction and Analysis:** Extract the 7 required properties from each generated graph.
2. **Selection Process:** Identify and return the graph that minimizes the **z-score normalized MAE**.

Calculation

For all graphs generated for the same textual description, each property is z-standardized using the formulas:

$$\text{normalized\_property}_i = \frac{y_i - \text{mean}_i}{\text{std}_i + \epsilon}$$

Next, the z-score MAE is calculated for each graph:

$$\text{mae\_st} = \frac{1}{n} \sum_{i=1}^{n} \left| \text{normalized\_gen}[i] - \text{normalized\_true}[i] \right|$$

Graph Generation

why to use z-score MAE instead of MAE

1. **Fair Comparison Across Properties:**
   - Properties like the number of nodes and clustering coefficient have different scales.
   - Normalization ensures evaluation on a common scale, avoiding disproportionate influence.

Graph Generation

$\longrightarrow$ This approach increased the overall computation time, as it required generating n graph samples for the same description.
$\longrightarrow$ Despite the increased time consumption, this method effectively reduced the public MAE error by half, improving it from 0.14016 to 0.07508.

What did not work

This project provided a valuable opportunity to experiment with multiple approaches. Although not all attempts were successful, each contributed to our learning experience. Below are the key methods and challenges:

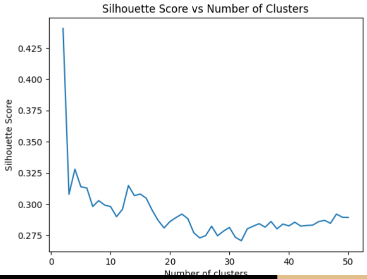
1. Implementing Encoders and Decoders: We explored architectures such as DCN, GCN, and GAT, tuning hyperparameters. Minimal improvement was observed, with the public MAE moving from 0.89274 (using GIN) to 0.88838. Adding text embeddings improved MAE slightly ($0.14604 \rightarrow 0.14540$), but computational cost outweighed the benefits.

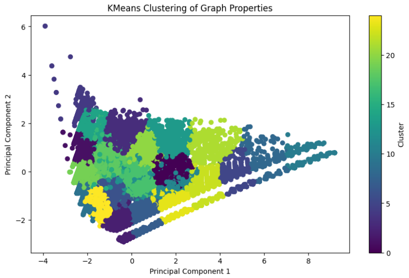
2. Graph Attention Networks (GAT): Experimented with GAT for varying node connection importance. Training was computationally expensive, taking over 5 hours for 78 epochs. Hardware limitations (CPU, RAM) further constrained this method.

What Did Not Work

Attempt: Incorporating Clustering Information

- To enhance the embeddings, we applied k-means clustering with an optimal number of clusters set to 25.
- For each graph, we created a one-hot encoded vector of size 25, indicating the cluster membership (1 for the corresponding cluster and 0 otherwise) and then concatenated with the existing embeddings to enrich the representation.





What Did Not Work

Outcome and Challenges

- The clustering approach did not improve the model's performance, possibly due to clusters being too compact to distinguish meaningful patterns in the data.
- To enhance clustering effectiveness, exploring **additional graph features** such as density and modularity might provide better results.

What can be tried

- Contrastive learning has shown success in creating robust and discriminative embeddings by learning to bring similar representations closer and push dissimilar ones apart. Techniques such as SimCLR or CLIP can be adapted to learn graph embeddings
- We can use an encoder and decoder with the attention mechanism, but it requires a more powerful machine than ours or With access to more advanced GPUs or distributed computing. When we tried to implement it, I had to reduce the model's complexity and halve the batch size. It also takes a whole day for compilation, but it's still an approach worth exploring further.

Conclusion

- In this competition, we had the chance to explore the intersection of text and graph data in a unique formulation, moving beyond conventional tasks like classification and regression.
- We started by diving into relevant literature to identify potential areas of improvement and to guide our experimentation.
- Throughout the process, we tested various approaches, learning what worked and what didn't along the way.
- In the end, our solution comprised an ensemble of different models to enhance the robustness of our predictions.

References I

Davide Rigoni, Nicolò Navarin, and Alessandro Sperduti.
Conditional constrained graph variational autoencoders for molecule design, 2020.