Conditional generation of graphs

Yassine OJ[1)], Dheeraj Parkash[2)], Antoine Peyronnet[3)]
[1)] ENSTA Paris, [2]Paris Saclay University , [3)]ENS Rennes-MVA

Summary

The objective of this project was to develop a generative model capable of producing graphs with specific structural properties, such as the number of nodes, edges, and connectivity. Our team's approach involved systematically exploring various families of generative models to identify the most suitable one for the task. Rather than relying solely on conventional methods, we introduced innovative hybrid models that combine different generative paradigms. Additionally, we incorporated advanced techniques, such as leveraging large language models (LLMs), to enhance the performance and quality of our results.

1 Conditional normalizing flows

Inspired by the example of NGG, the team build a hybrid model composed of an autoencoder that provides latent representations of the graph and a normalizing flow that processes in the same latent space. The autoencoder used in our model is an approximation of the one used in the NGG. In fact, we made the choice of using a deterministic autoencoder. The condition vector on the graph is introduced in the block of the normalizing flow, hence the use of conditional normalizing flows.

1.1 Definition of normalizing flows

Normalizing flows are a class of generative models that define a complex probability distribution $p_X(x)$ by transforming a simple base distribution $p_Z(z)$ (e.g., Gaussian) through a series of invertible and differentiable transformations $f = f_1 \circ f_2 \circ \cdots \circ f_K$. Using the change of variables formula, the resulting density is computed as:

$$p_X(x) = p_Z(z) \left| \det \left(\frac{\partial z}{\partial x} \right) \right|,$$

where $z = f^{-1}(x)$, and $\det \left(\frac{\partial z}{\partial x}\right)$ is the determinant of the Jacobian matrix of the transformation f.

In our case we used the **Conditional Normalizing Flows**. They are an extention to normalizing flows to model conditional probability distributions $p_X(x \mid c)$, where c represents additional conditioning information. The transformation f becomes condition-dependent, $f_c = f_{c,1} \circ f_{c,2} \circ \cdots \circ f_{c,K}$, and the base distribution $p_Z(z)$ may also depend on c. Using the change of variables formula, the conditional density is expressed as:

$$p_X(x \mid c) = p_Z(z \mid c) \left| \det \left(\frac{\partial z}{\partial x} \right) \right|,$$

where $z = f_c^{-1}(x)$, and $\det \left(\frac{\partial z}{\partial x}\right)$ is the determinant of the Jacobian matrix of the conditional transformation f_c.

1.2 Affine Coupling Layers:

One of the main challenges of normalizing flows lies in designing the function f, which must be invertible, have an efficiently computable inverse, and allow for efficient computation of $\log \det \left| \frac{\partial f}{\partial z} \right|$.

Affine coupling layers are one family of those functions. They are a type of coupling layer used in normalizing flows, which transforms data in a way that guarantees the bijectiveness of the transformation while maintaining computational efficiency. In an affine coupling layer, the transformation $f(x)$ is split into two parts:

$$f(x) = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \exp(s(x_1)) \end{pmatrix} + \begin{pmatrix} 0 \\ t(x_1) \end{pmatrix}$$

The inverse of the transformation is also an affine transformation, making the transformation bijective. The Jacobian determinant for this layer is easy to compute and is given by:

$$\det \left(\frac{\partial f(x)}{\partial x} \right) = \exp(s(x_1))$$

This structure enables efficient training and inversion of the flow.

1.3 Conditional Modules

We incorporate the condition into our model in a manner similar to the approach described in [C. Winkler et al., 2019]. The conditioning is introduced in the prior and affine coupling modules. For the prior, we set the mean and variance as functions of x. For the affine coupling layers, we pass x to the scale and translation networks so that:

Conditional Prior: $p(z|x) = \mathcal{N}(z; \mu(x), \sigma^2(x))$

Conditional Coupling: $y_0 = s(z_1, x) \cdot z_0 + t(z_1, x); \quad y_1 = z_1$

In practice, these functions are implemented using deep neural networks. First, the conditioning term x is transformed into a rich representation $h = g(x)$ using a large network g. Subsequently, each function in the flow is applied to a concatenation $[\cdot, \cdot]$ of h and the relevant part of z. For example, the translation of a conditional coupling is computed as follows:

$$t(z_1, x) = \mathrm{NN}([z_1, h])$$

1.4 Stabilizing the model

One of the problems we faced in this model is the explosion of scaling factors. It was a normal result given the existence of the exponential in the scale function. [Daniel Andrade, 2024] proposes a solution to this problem by placing an upper bound on the scaling factors $\exp(s(z_1, x))$.

The idea is to threshold high and low scaling factors to ensure that $\alpha_{\text{neg}} \leq s(z_1, x) \leq \alpha_{\text{pos}}$, where the thresholds α_{pos} and α_{neg} is set to a fixed small value. The goal of is to avoid exploding scaling factors as mentioned before. The importance of is to ensure the stable calculation of the inverse f^{-1}, but insuring ensuring that $\exp(s(z_1, x))$ does not get too close to zero.

We therefore place an additional threshold that enforces $s(z_1, x) \geq -\alpha_{\text{neg}}$. Rather than hard thresholding, we found empirically that the following asymmetric soft clamping improved training:

$$c(s) = \begin{cases} \frac{2}{\pi} \alpha_{\text{pos}} \arctan\left(\frac{s}{\alpha_{\text{pos}}}\right), & \text{if } s \geq 0, \\ \frac{2}{\pi} \alpha_{\text{neg}} \arctan\left(\frac{s}{\alpha_{\text{neg}}}\right), & \text{if } s < 0, \end{cases}$$

1.5 Mutual Information Neural Estimator (MINE)

After several tries, the model did not have a good performance. To explain this, we made the hypothesis that the latent representation generated by the normalizing flow model is not very representative of the condition vector. Therefore we used Mutual Information Neural Estimator (MINE) to create more link between the condition and the latent representation.

Mutual Information Neural Estimator (MINE) is a method for estimating the mutual information $I(X;Y)$ between two random variables X and Y. MINE relies on the Donsker-Varadhan representation of the Kullback-Leibler divergence:

$$I(X;Y) = \sup_{T_\theta} \mathbb{E}_{p(X,Y)}[T_\theta(x,y)] - \log \mathbb{E}_{p(X)p(Y)}\left[e^{T_\theta(x,y)}\right],$$

where T_θ is a neural network parameterized by θ, which acts as a critic function to distinguish between joint samples $(x, y) \sim p(X, Y)$ and marginal samples $(x, y) \sim p(X)p(Y)$.

MINE is trained by maximizing the above objective using gradient-based optimization. In practice, the expectations are approximated using Monte Carlo sampling:

$$I(X;Y) \approx \frac{1}{N} \sum_{i=1}^{N} T_\theta(x_i, y_i) - \log\left(\frac{1}{N} \sum_{i=1}^{N} e^{T_\theta(x_i, y_i)}\right),$$

where (x_i, y_i) are samples from the joint distribution $p(X, Y)$, and (x_i, y_i) are permuted samples to approximate the marginal distribution $p(X)p(Y)$.

MINE was introduced to the loss of the normalizing flow model. Therefore, the loss of the model becomes te addition of the normalizing flow loss and the $-I(X;Y)$ multiplied by a hyper parameter β.

1.6 Results and conclusion

The results of this model were not good. At the base case, the result on the test dataset was : **0.98**. We explain this by the fact that the latent embeddings produced by the normalizing flow were not very representative of the condition despite all the different techniques used for this aim.

2 LLMs

In this study, embeddings generated from the pretrained T5 and BERT models were used to condition a Variational Graph Autoencoder (VGAE) model for improved link prediction. Initially, when the embeddings from these models were applied to the baseline VGAE, the performance was evaluated using the Mean Absolute Error (MAE), which was around 0.8. The baseline model utilized the standard F1 loss function during training. However, two key modifications were introduced to improve the model's performance: first, the F1 loss was replaced with the sum of the L1 loss; second, L2 regularization was added to the loss function. These changes led to a significant improvement in the model's ability to learn from the graph data.

When training the VGAE pipeline with T5 embeddings, the MAE dropped to 0.41, indicating a substantial improvement in the model's predictive accuracy. Similarly, when BERT embeddings were used, the model achieved an MAE of 0.50, which, although slightly higher than with T5 embeddings, still represented a clear enhancement over the baseline model.

These results underscore the importance of leveraging transformer-based models such as T5 and BERT for generating high-quality text embeddings that condition a graph-based model like VGAE.

The use of embeddings from T5 and BERT in the VGAE model highlights the significance of incorporating pre-trained models for text data, particularly for tasks like link prediction and graph analysis. T5 and BERT are capable of capturing rich contextual information from text, which is crucial when learning graph structures that depend on node features. The reduction of F1 loss to a sum form, along with the addition of L2 regularization, has proven to be an effective strategy for improving model performance, further validating the utility of these embedding techniques in enhancing graph-based deep learning models.

3 Graph Transformer

The **Graph Transformer** model utilized in this work represents a novel approach that integrates the power of transformer-based architectures with graph-based data processing. Traditionally, transformers excel at handling sequential data through self-attention mechanisms [7], while Graph Neural Networks (GNNs) focus on learning from graph structures by passing messages between neighboring nodes [8]. The **Graph Transformer** combines these strengths, utilizing a transformer-based encoder to learn node representations and a decoder to reconstruct the graph structure, effectively leveraging global attention in graph data.

The **Graph Transformer Encoder** begins by projecting input node features $x \in \mathbb{R}^{N \times D}$ into a hidden space using a linear transformation:

$$x' = W_{\text{proj}} \cdot x$$

where $W_{\text{proj}} \in \mathbb{R}^{D' \times D}$ is the projection matrix, and D and D' represent the input and hidden dimensions, respectively. The model then applies multiple transformer convolution layers, each with self-attention mechanisms that operate on the graph's adjacency structure. The self-attention mechanism at layer l is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q, K, V represent the query, key, and value matrices, respectively, and d_k is the dimensionality of the query and key [7]. After the encoding phase, a global pooling operation aggregates node features to create a fixed-size graph representation, z:

$$z = \text{GlobalPool}(x)$$

This representation is further passed through a fully connected layer to obtain the latent vector z_{latent}, which encodes the graph's overall structure [9]. The **Graph Transformer Decoder** uses this latent representation to reconstruct the adjacency matrix $\hat{A}$, predicted as:

$$\hat{A} = \text{Decoder}(z_{\text{latent}})$$

In this work, **textual description embeddings**, generated using pre-trained models like T5[10] and BERT [11], were incorporated into the graph transformer pipeline to condition the model's learning. These embeddings serve as additional input features to the graph, enriching the node representations with semantic information derived from text. The model was trained to predict graph edges, and the **Mean Absolute Error (MAE)** was evaluated as a performance metric. The loss function combines a reconstruction loss, measured by mean squared error (MSE) between the predicted and actual adjacency matrices $\hat{A}$ and A, and a Kullback-Leibler divergence (KLD) loss:

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \beta \mathcal{L}_{\text{KLD}}$$

where:

$$\mathcal{L}_{\text{recon}} = \frac{1}{N}\sum_{i=1}^{N} \|A_i - \hat{A}_i\|_2^2$$

$$\mathcal{L}_{\text{KLD}} = -0.5 \sum \left(1 + \log(\sigma^2) - \mu^2 - \sigma^2\right)$$

where μ and σ^2 are the mean and variance from the latent distribution, and β is a hyperparameter controlling the importance of the KLD term[12].

When conditioned on T5 embeddings, the model achieved a **MAE of approximately 0.70**, indicating promising results. This suggests that incorporating textual embeddings into graph structures significantly enhances the model's ability to predict graph relationships, demonstrating the potential of combining language and graph-based models for complex tasks.

4 Linear autoencoder model

We began with a straightforward approach by designing a simple Encoder-Decoder architecture, relying exclusively on linear layers and normalization layers. The model took as input seven features (e.g., nodes, edge indices, edges, etc.) and predicted the graph structure.

Using just four linear layers paired with batch normalization in both the encoder and decoder each layer having approximatively 500 neurons, activated by Leaky ReLU functions, we achieved a strong performance. The encoder compressed the input into a latent space of 200 neurons, while the decoder expanded it back. By carefully tuning hyperparameters, such as the dropout rate, learning rate, and leveraging a regularized L1 loss function, the model reached a score of 0.28.

This led the way to think about using VAE-Diffusion Model.[GenGraph] [NeuralGraphGen]

5 Using VAE-Diffusion Model

VAEs are generative models that learn latent representations by combining an encoder and a decoder. They balance reconstruction loss and a regularization term based on the Kullback-Leibler (KL) divergence to encourage smooth latent spaces.

The encoder maps the input into the latent space. We chose a VAE for generating structured graph outputs. The role of the VAE is to encode the input into a Gaussian probabilistic distribution defined by its two parameters $(\mu, \log \text{var})$, resulting in a compact latent space. Learning from this latent space facilitates smooth graph reconstructions.

Encoder Design: We implemented a Graph Isomorphism Network (GIN) as our encoder because of its ability to capture graph-level and node-level embeddings with high expressiveness. Key design choices include:

- **LeakyReLU Activations:** Mitigates the issue of dying neurons by allowing small gradients for negative values, enhancing learning stability.
- **Batch Normalization:** Ensures stable training by normalizing layer outputs, addressing the exploding/vanishing gradient problem.

The latent space compactly encodes the essence of input graphs, with smoothness ensured by the KL divergence regularization term. Probabilistic sampling enables the generation of diverse graph structures. **Decoder Design:** The decoder reconstructs the input graph (adjacency matrix) from the latent representation. It uses:

1. **Multilayer Perceptrons (MLPs):** Capture complex relationships between latent features.
2. **Gumbel-Softmax:** Facilitates sampling of discrete adjacency structures while maintaining differentiability during backpropagation.

The decoder learns a probability distribution over adjacency matrices, sampled to produce discrete graph structures closely matching the original input graphs.

Losses and Preventing Overfitting

We experimented with various reconstruction losses: L1 mean, L1 sum, L2 Frobenius, and Binary Cross Entropy. The best performance was achieved with a regularized L1 sum loss:

$$L_{\text{total}} = \sum_{i,j} \left| A_{i,j} - \hat{A}_{i,j} \right| + \beta L_{\text{KL}} + \lambda \|A - \hat{A}\|_F^2$$

This formulation allows aggregation of errors across all elements of the matrix:

$$L_{\text{recon}} = \sum_{i,j} \left| A_{i,j} - \hat{A}_{i,j} \right|$$

The L1 sum loss captures edge reconstruction errors and scales well for both large and small graphs, avoiding bias due to graph size. To prevent overfitting, we introduced:

- **KL Regularization:** Structures the latent space, enabling meaningful generation.
- **L2 Regularization:** Ensures smooth reconstructions.
- **Dropout:** Increased from 0.2 to 0.3 for better generalization.

Denoiser

We used a diffusion model as the denoiser, comprising two components:

1. **Forward Diffusion:** Adds Gaussian noise to the input graph at each timestep using the q-sample function.
2. **Reverse Diffusion:** Removes noise iteratively using the p-sample function guided by a neural network (`DenoiseNN`).

Forward Diffusion: The forward process is mathematically defined as:

$$x_t = \sqrt{\alpha_{\text{cumprod}_t}} \cdot x_0 + \sqrt{1 - \alpha_{\text{cumprod}_t}} \cdot \epsilon$$

where $\epsilon \sim \mathcal{N}(0, I)$ represents Gaussian noise. **Reverse Diffusion:** The `DenoiseNN` predicts the noise $\hat{\epsilon}_t$ added to the representation at timestep t. It uses:

- **Sinusoidal Position Embeddings:** Encodes timestep t for temporal awareness.
- **Conditional Embeddings:** Integrates graph-specific properties via an MLP.
- **MLP Layers:** Learns mappings from noisy inputs to denoised outputs.

The loss function minimizes the difference between true noise and predicted noise $(\epsilon - \hat{\epsilon}_t)$.

Results

We focused on improving robustness through loss function optimization. A regularized L1 sum loss for the VAE and a combined loss for the denoiser proved most effective. We attained a MAE score of 0.1779 with that model. We put the parameters in the tables underneath [1]

Key observations:

- Keeping the same number of encoder and decoder layers in the VAE yielded better performance.
- Adding MLP layers in the denoiser did not improve results.

[1] See the vae_diffusion.ipynb folder

Hyperparameter	Value
lr (VAE)	8×10^{-4}
dropout (VAE)	0.3
batch_size (VAE)	256
epochs_autoencoder (VAE)	200
hidden_dim_encoder (VAE)	64
hidden_dim_decoder (VAE)	256
latent_dim (VAE)	32
n_max_nodes	50
n_layers_encoder (VAE)	4
n_layers_decoder (VAE)	4

Hyperparameter Diffusion	Value
spectral_emb_dim	10
epochs_denoise (denoiser)	150
timesteps	500
hidden_dim_denoise (denoiser)	512
n_layers_denoise (denoiser)	3
train_autoencoder	True
train_denoiser	True
dim_condition	128
n_condition	7

Table 1: Final Hyperparameters for our VAE-diffusion model

6 Overall Conclusion

We proposed a model to predict graphs from literal descriptions, exploring three approaches: normalizing flows, Transformers and LLMs, and a VAE-diffusion framework. Among these, the VAE-diffusion model demonstrated superior performance, achieving the best Mean Absolute Error (MAE) score. By prioritizing robustness through architectural design, regularization, and hyperparameter optimization, the VAE-diffusion approach effectively captured complex graph structures.

These findings highlight the potential of probabilistic and iterative methods for graph generation, paving the way for future improvements through hybrid models and broader datasets.

References

Christina Winkler, Daniel Worrall, Emiel Hoogeboom, Max Welling (2019). Learning Likelihoods with conditional normalizing flows, arXiv:1912.00042

Daniel Andrade (2024). Stabilizing training of affine coupling layers for high-dimensional variational inference. Machine Learning: Science and Technology, Volume 5, Number 4

Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, R Devon Hjelm (2018). MINE: Mutual Information Neural Estimation, arXiv:1801.04062 MINE: Mutual Information Neural Estimation. *arXiv:1801.04062*, 2018.

Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, Kevin Swersky (2020). Graph Normalizing Flows. *arXiv preprint arXiv:2006.00951.*

Christos.X and Iakovos Evdaimon. Generating Graphs with Specified Properties. *Kaggle Competition*, 2024.

Iakovos Evdaimon, Giannis Nikolentzos, Christos Xypolopoulos, Ahmed Kammoun, Michail Chatzianastasis, Hadi Abdine, and Michalis Vazirgiannis. Neural Graph Generator: Feature-Conditioned Graph Generation using Latent Diffusion Models. *arXiv:2403.01535*, 2024.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. A., Kaiser, Ł., Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Kipf, T. N., Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907.*

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y. (2018). Graph attention networks. *arXiv preprint arXiv:1710.10903.*

Raffel, C., Shinn, C., Roberts, A., Lee, S., Narang, S., Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 1-67.

Devlin, J., Chang, M. W., Lee, K., Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT 2019.*

406 Kingma, D. P., Welling, M. (2013). Auto-encoding
407 variational bayes. *arXiv preprint arXiv:1312.6114*.