Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Vector-Quantized Generative Adversarial Networks for Graphs

Master's Thesis

Güney Işık Tombak

gtombak@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Mr. Karolis Martinkus
Dr. Nathanaël Perraudin
Prof. Dr. Roger Wattenhofer

September 12, 2022

Acknowledgements

First of all, I would like to express my deepest gratitude to Mr. Karolis Martinkus, Dr. Nathanaël Perraudin, and Prof. Roger Wattenhofer for granting me the privilege of working on such an intellectually fulfilling project. Their guidance and support throughout this study were invaluable. Whenever I had a question, they introduced me to new perspectives and helped me find novel understandings. Thanks to them, I learnt a lot throughout this project.

I will always be appreciative to my family for their unwavering love and assistance throughout my life. Because of their support and belief in me, I was able to persevere through the most difficult circumstances.

Last but not least, I want to thank my friends for being by my side through thick and thin. I particularly want to express my gratitude to Sofi Farazande, the apple of my eye, for her unfailing support, love, and encouragement.

Abstract

Generative models for graph-structured data have various use cases in real-life applications such as chemical synthesis, circuit design, and telecommunications. As a novel approach, we introduce a generative framework for graphs by combining vector-quantized variational autoencoders (VQ-VAEs) and generative adversarial networks (GANs). The reasoning behind this is to learn an embedding space in which the graph generation process is hopefully easier. In a greedy fashion, we train the VQ-VAE first and use learned posterior categorical distributions for vertices as embedding space during the training of the GAN. Our model has inherently desirable characteristics of permutation equivariance on graph vertices and one-shot generation capability. However, the experiments on three types of well-known graph types show that the model is insufficient for realistic graph generation. We discuss the possible problems in the framework and the means of improvement for future work.

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Autoencoder
BN	Batch Normalization
Conv	Convolutional
FC	Fully Connected
FFN	Feed Forward Network
FN	Number of False Negatives
FP	Number of False Positives
GAN	Generative Adversarial Network
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GIN	Graph Isomorphism Network
GNN	Graph Neural Network
LN	Layer Normalization
MHA	Multi Head Attention
MLP	Multi Layer Perceptron
MMD	Maximum Mean Discrepancy
ReLU	Rectified Linear Unit
RKHS	Reproducing Kernel Hilbert Space
S2G	Set to Graph
SBM	Stochastic Block Model
TE	Test
TN	Number of True Negatives
TP	Number of True Positives
TR	Training
TV	Total Variation
VAE	Variational Autoencoder
VL	Validation
VQ-VAE	Vector Quantized Variational Autoencoder
WGAN	Wasserstein Generative Adversarial Network

List of Symbols

$\odot$	Element-wise Multiplication
$\|\|$	Concatenation
$\mathbf{A}$	Adjacency Matrix
B_ζ	VQ-VAE Bottleneck
$\mathcal{C}$	VQ-VAE Codebook
D_θ	Decoder
$\mathcal{E}$	Edge Set
E_ϕ	Encoder
F_ψ	Discriminator
$\mathcal{G}$	Graph
G_γ	Generator
L	Loss
$\boldsymbol{\ell}$	Logits
$\bar{\mathcal{L}}$	Normalized Laplacian
$\mathbf{M}$	Adjacency Masking Matrix
$\mathbf{m}$	Node Masking Vector
N	Number of Nodes
$\mathcal{P}$	Pooling Functions Set
$\mathbf{q}$	Posterior Categorical Distribution
$\mathcal{V}$	Vertex Set
v	Graph Vertex / Node
$\mathbf{X}$	Node Embeddings

α	Cooling Rate
β_D	Broadcast Part of Decoder
Γ_D	Graph to Graph Part of Decoder
λ_{gp}	WGAN Gradient Penalty Scale
λ_{kl}	VQ-VAE KL Divergence Scale
$\boldsymbol{\lambda}(\mathbf{K})$	Eigenvalues of Matrix K
π	Permutation / Ordering
ρ	ReLU
Σ_D	Set to Set Part of Decoder
σ	Sigmoid
σ_{gsm}	Gumbel-Softmax
σ_{max}	Softmax
τ	Temperature for Gumbel-Softmax

Chapter 1

Introduction

1.1 Graph Preliminaries

In this project, we work on a new paradigm for the graph generation. Formally, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of two main sets of elements: a set of vertices $\mathcal{V} = \{v_i\}_{i=1}^{N}$ and a set of edges $\mathcal{E}$. A vertex constitutes the main element of a graph, which is also called as a node, while an edge represents a connection between two nodes: $\mathcal{E} = \{e_{i,j} = (v_i, v_j) | v_i, v_j \in V\}$.

The classical way to represent the edge relations in a graph is using an adjacency matrix $\mathbf{A}$, where the i, j^{th} entry of the matrix defined whether there is an edge from vertex v_i to vertex v_j or not. This matrix representation comes with a requirement of node ordering π that maps nodes to rows and columns. Naturally, we can define $N!$ possible node orderings and we could define all possible orderings as a set of permutations Π.

In its most general form of graphs, which is known as a weighted directed graph, the edges are defined independently from each other by weight $w_{i,j}$ and a missing edge is defined by 0. Whereas for our work, we focus on only unweighted undirected graphs, where the edges are defined by their presence only, all of them are symmetric, and self-loops are forbidden. In this setting, we can define an adjacency matrix under a node ordering $\mathbf{A}^{\pi} \in \mathbb{R}^{N \times N}$ formally as

$$\mathbf{A}^{\pi}_{i,j} = \mathbb{1}[(\pi(v_i), \pi(v_j)) \in \mathcal{E}] \tag{1.1}$$

such that $\mathbf{A}^{\pi}_{i,j} = \mathbf{A}^{\pi}_{j,i}$ and $\mathbf{A}^{\pi}_{i,i} = 0$.

1.2 Problem Definition

The purpose of constructing a generative model for graphs is to learn a distribution over graphs based on a set of observed instances $\mathbb{G} = \{\mathcal{G}_i\}_{i=1}^{s}$ taken from a predefined data distribution $p(\mathcal{G})$. Such models are useful in real life applications and various research areas such as chemical design [1] [2] [3], network science [4],

and circuit design [5]. For the assessment of a graph generative model, we can consider the following properties [6]:

1. **Permutation Equivariance:** a graph with N number of nodes could be described by $N!$ orderings. A model should consider each ordering with the same probability for a particular graph, i.e.: $p_{\Pi}(\pi_i) = \frac{1}{N!}$ for $i = 1, 2, ..., N!$. For a function f, the permutation equivariance property on a set of vertices $\mathcal{V}$ is satisfied if

$$f(\pi_i(\mathcal{V})) = f(\pi_j(\mathcal{V})) \text{ for } i, j = 1, 2, ..., N!. \tag{1.2}$$

2. **Scalability:** a model should be able to handle the type of graph distribution with any number of nodes. Moreover, it also should be able to handle a large number of graphs in a reasonable time span with an achievable computational complexity.

3. **Expressive Power:** a model should correctly learn about the structural details and patterns, represented both locally and globally in graphs.

4. **Novelty:** a model should be able to generate graphs that are not observed in the training set.

Although the above properties are difficult to achieve together, we fulfil the first two by using a permutation equivariant model which generates graphs in one-shot manner. Overall, we propose a generative model for graphs by combining vector-quantized variational autoencoders (VQ-VAEs) and generative adversarial networks (GANs). The model is trained in a greedy fashion by training the VQ-VAE first on the graph reconstruction task and then the GAN for generation in the latent space of the VQ-VAE.

The remainder of the thesis is organized as follows: Chapter 2 consists of a short literature survey regarding generative approaches for graphs. In Chapter 3, our proposed framework and training procedure are explained in detail. We present our findings by evaluating our proposed framework on three graph datasets in Chapter 4. Finally, we discuss possible problems with the framework and the means of improvement in Chapter 5.

Chapter 2

Related Work

2.1 Traditional Approaches

The early approaches for graph generation are mathematical models. As one of the firsts, Erdős–Rényi–Gilbert model [7] [8] is based on a statistical analysis of graphs which constituted the founding stones of random graph theory. In many years, random graph models have been studied in network science [9], social sciences [10], and physics [11]. Although these approaches are fundamental in our understanding of graph generation, they lack the expressive power and novelty required for real-life applications.

2.2 Recent Approaches

Thanks to the recent progress in data-driven deep learning models, deep generative solutions have been successfully implemented for a wide range of tasks in various domains such as computer vision, natural language processing, and industrial design [12]. With the advance in deep learning models for graph data with the introduction of the graph neural networks (GNNs) [13] and specific GNN architectures such as graph convolutional networks (GCNs) [14], and graph attention networks (GATs) [15]; researchers can apply the deep learning frameworks to achieve state-of-the-art results in graph generation.

In this sense, two possible classifications of these methods are as follows: likelihood-based vs. implicit generative and autoregressive vs. one-shot.

2.2.1 Likelihood-Based vs. Implicit Generative Methods

We can compare likelihood-based and implicit generative approaches with their respective successful representatives in today's deep learning research: variational autoencoders [16] (VAEs) and generative adversarial networks (GANs) [17], respectively. Briefly, the former aims to learn the latent representation of a dataset

via reconstruction of the true data while the latter proposes new samples and discriminates them from the true distribution in an adversarial framework to learn the data distribution. One important advantage of VAEs over GANs is their stability and convenience during training [18]. In general, Graph VAEs are able to learn the latent representation of small graphs such as molecules [19] [20] [3] [21] but they are mostly insufficient for large graph synthesis. On the contrary, it is known that implicit generation stimulates higher diversity, which results in better performance in large graph generation [22]. In consequence, GANs have been used with various settings and configurations in the last few years for graph generation [23] [24] [25] [26] [6].

In our work, we employ both a likelihood-based and an implicit approach to graph generation. We use a vector quantized VAE (VQ-VAE) to learn the latent representation of the nodes and then used a GAN in the latent space to generate new samples.

2.2.2 Autoregressive vs. One-shot Methods

Mathematically we can define the autoregressive graph generation process as below

$$p(\mathcal{G}) = \prod_{i}^{N} p(v_i, \mathcal{E}(v_i)|\mathcal{V}_{<i}, \mathcal{E}(\mathcal{V}_{<i})) \tag{2.1}$$

where each node and its connections are generated considering the nodes and edges already generated for that particular graph [2] [6] [27]. Due to the additive nature of this iterative process, it circumvents the scalability issues that coincided with fixed-size methods. Moreover, such a structure allows for the better management of the generation process regarding its divide et impera approach. One major flaw of this process is the requirement of an order on the nodes, which makes the generation process permutation sensitive. Although there are various ways to alleviate the issue such as using canonical orderings according to the graph structure, there is no stable canonical ordering. Moreover, the iterativeness of the process makes both training and inference times longer and unable to use computation parallelization techniques which induces scalability problems eventually.

Learning to generate all edges in one step circumvents both the scalability issues and the requirement of an order on the nodes. Such methods are known as one-shot. Most of the likelihood-based [19] [20] [21] and implicit methods [23] [24] [25] [6] discussed in Section 2.2.1 are one-shot approaches. Although one-shot is mathematically more pleasing, in general, it requires more elaboration on the model and the training process. In our work, we also use a one-shot approach to generate graphs.

2.2.3 Other Methods

Other than the above categorizations, we can discuss unconventional approaches to graph generation. One of them is using the graph normalizing flows [28], where a permutation-invariant reversible graph neural network with small memory requirements is proposed. Another method focuses on a specific loss function based on annealed Langevin dynamics [29], which is known for its use in the mathematical modelling of molecular systems. We can also consider NetGAN [25], a random walk-based generative model for molecular graphs as an interesting approach for the task at hand.

Chapter 3

Methodology

3.1 Model

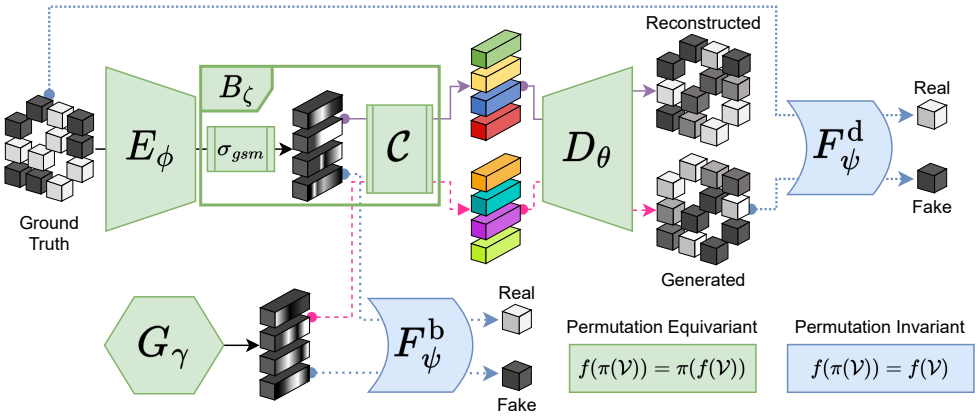


Figure 3.1: Overall architecture of our model.

To generate novel graphs based on a set of graphs with particular characteristics, we follow a node embedding-based approach and use two well-known generative deep learning frameworks: VQ-VAEs [30] and GANs [17]. Inspired by Taming Transformers by Esser et al. [31], the main idea is first teaching a node-embedding autoencoder to reconstruct the adjacency matrices and then training a GAN on node embeddings to generate novel graphs by decoding using the pretrained autoencoder. We use posterior categorical distributions $\mathbf{q}$ of the VQ-VAE as the node embeddings to employ the power of vector quantization.

As described in Fig. 3.1, we use two different configurations regarding the input of the discriminator: a bottleneck discriminator (F_ψ^{b}) which takes node embeddings and a decoder discriminator (F_ψ^{d}) which takes adjacency matrices as input, each configuration using only one of them. The encoder (E_ϕ), bottleneck (B_ζ), decoder (D_θ), and generator (G_γ) models are permutation equivariant on nodes whereas both of the discriminator (F_ψ) models are permutation invariant.

3.1.1 Autoencoder

First introduced as a nonlinear principal component analysis method by Kramer [32] in 1991, an autoencoder aims to learn efficient and possibly informative codings of unlabeled data by constructing nonlinear features. The architecture consists of two parametrically defined components: an encoder (E_ϕ) and a decoder (D_θ), which are mappings from the data space ($\mathcal{X}$) to encoding space ($\mathcal{Z}$) and from the encoding space ($\mathcal{Z}$) to the reconstruction space ($\hat{\mathcal{X}}$), respectively.

In its most general sense, the model parameters are tuned such that

$$\phi^*, \theta^* = \underset{\phi\ ,\ \theta}{\operatorname{argmin}} \sum_{\mathbf{x}\in\mathcal{X}} \|\mathbf{x} - D_\theta(E_\phi(\mathbf{x}))\|. \tag{3.1}$$

One of the main issues with autoencoders for graphs is their permutation dependency: they learn and encode the graphs with the ordering of the nodes in the input adjacency matrices. To avoid this problem, we embed the nodes as independent vectors in the latent space using a permutation invariant encoder and decoder. In this setting, for an undirected graph of N nodes, the data, latent, and reconstruction space can be defined as

$$\mathcal{X} = \{0,1\}^{\frac{N(N-1)}{2}} \quad , \quad \hat{\mathcal{X}} = [0,1]^{\frac{N(N-1)}{2}} \quad , \quad \mathcal{Z} = \mathbb{R}^{N\times d_z} \tag{3.2}$$

where d_z is the latent space dimension and in order to achieve differentiability for backpropagation, the reconstruction space is defined continuously as the natural mapping from the original discrete one.

Encoder

Regarding the permutation equivariance requirement, we use a Graph Isomorphism Network (GIN) [33] in the encoder. It is a cascaded network of unit structures using convolution and fully connected layers:

$$\mathbf{x}'_i = \mathrm{MLP}\left(\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} \mathbf{x}_j\right) \tag{3.3}$$

where $\mathcal{N}(i)$ is a set of indices of vertices neighbour to vertex i. This relation can be represented in the matrix form as

$$\mathbf{X}' = \mathrm{MLP}\left((\mathbf{A} + \mathbf{I}) \cdot \mathbf{X}\right) \tag{3.4}$$

which is a GIN convolution succeeded by a node-wise multi-layer perceptron (MLP). For the initial node embeddings, we used a vector consisting of ones and zeros for masked and non-masked nodes, respectively. Using always the same output dimension per node, the final output is computed as the sum of all unit outputs. A detailed representation of the encoder architecture is visualized in Fig. 3.2.

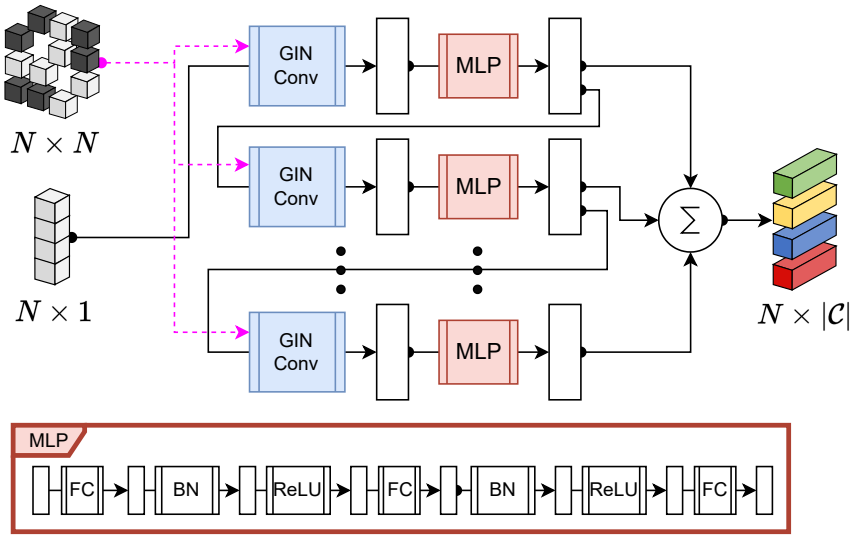


Figure 3.2: Architecture of the encoder.

Decoder

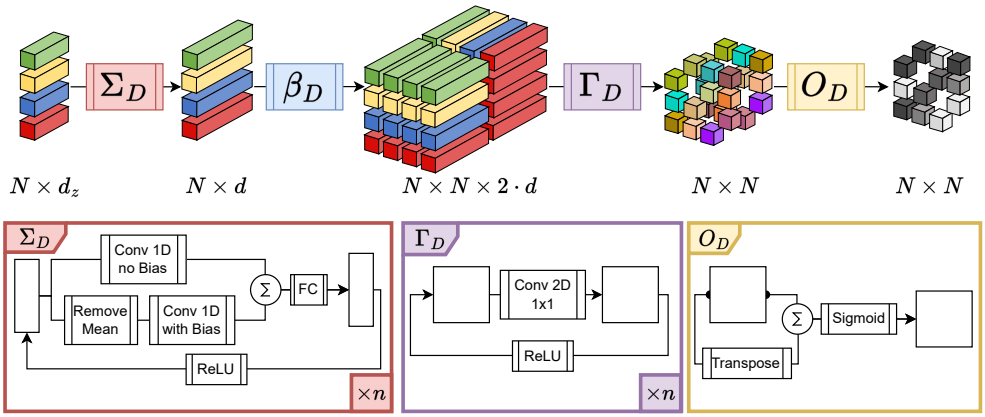


Figure 3.3: Architecture of the decoder.

For the decoder part of the autoencoder, we use Set2Graph (S2G) [34] model regarding its permutation equivariance on graph vertices. It consists of three networks: $\tilde{D}_\theta = \Sigma_D \circ \beta_D \circ \Gamma_D$ where Σ_D is a set-to-set network applied to the node embeddings, β_D is a non-learnable broadcasting set-to-graph layer, and Γ_D is a graph-to-graph network.

In addition to these three networks, we also use a non-learnable output layer O_D that symmetrizes and normalizes the elements of the reconstructed adjacency

matrix as well as sets diagonals to zero

$$O_D(A) = \texttt{diag}_0\left(\sigma\left(\frac{\mathbf{A}+\mathbf{A}^\top}{2}\right)\right) \tag{3.5}$$

where $\sigma(x) = (1+\exp(-x))^{-1}$ is a sigmoid and $\texttt{diag}_0$ sets diagonal elements of the matrix to zero. A detailed description of the decoder architecture can be found in Fig. 3.3.

For the training of the vanilla autoencoder, we use two types of binary cross entropy losses, namely standard and normalized cross entropy loss:

$$L^{\text{AE}}_{\phi,\theta}(\mathcal{E}) = \sum_{(v_i,v_j)\in\mathcal{E}} log(\mathbf{A}_{i,j}) \ , \ L^{\text{AE}}_{\phi,\theta}(\mathcal{E}') = \sum_{(v_i,v_j)\notin\mathcal{E}} log(1-\mathbf{A}_{i,j}) \tag{3.6}$$

$$L^{\text{AE}_{\text{standard}}}_{\phi,\theta} = \frac{L^{\text{AE}}_{\phi,\theta}(\mathcal{E}) + L^{\text{AE}}_{\phi,\theta}(\mathcal{E}')}{|\mathcal{E}| + |\mathcal{E}'|} \tag{3.7}$$

$$L^{\text{AE}_{\text{normalized}}}_{\phi,\theta} = \frac{|\mathcal{E}'|L^{\text{AE}}_{\phi,\theta}(\mathcal{E}) + |\mathcal{E}|L^{\text{AE}}_{\phi,\theta}(\mathcal{E}')}{2|\mathcal{E}||\mathcal{E}'|} \tag{3.8}$$

where $\mathcal{E}$ and $\mathcal{E}'$ are the set of matrix elements in the original adjacency matrices describing an edge or no edge, respectively. Details about the use of these loss functions can be found in Section 3.2.1.

Bottleneck

Although the vanilla autoencoder is a good backbone for embedding graph vertices, we require a regularized latent space with generative capabilities for the GAN part. To achieve these properties, we use a vector quantized bottleneck layer (B_ζ) [30] that quantizes the latent space into a one hot encoding using a codebook ($\mathcal{C}$) of latent vector elements. These encodings are known as posterior categorical distributions ($\mathbf{q}$). Such mapping from the encoder outputs, which are also called logits ($\boldsymbol{\ell}$), to a probability space can be achieved by using a softmax activation function. To have a continuous backpropagation during the training, the sampling process is a weighted sum such that

$$\mathbf{z} = \sum_{i=1}^{|\mathcal{C}|} \mathbf{q}_i \cdot \mathcal{C}_i \tag{3.9}$$

while during the inference, it is changed to an argmax operation

$$\mathbf{z} = \mathcal{C}_{i^*} \ , \ i^* = \arg\max_i \mathbf{q}_i. \tag{3.10}$$

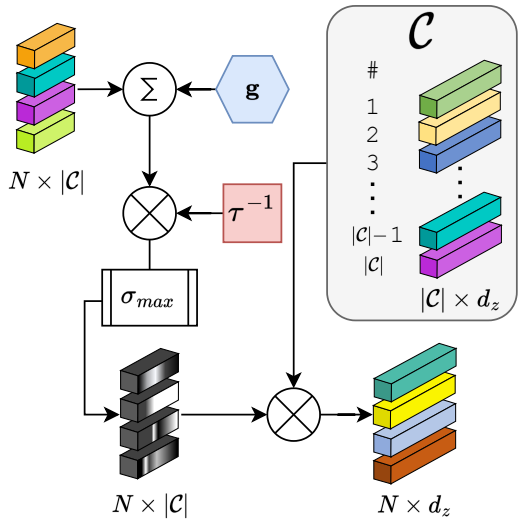


Figure 3.4: Bottleneck structure using codebook $\mathcal{C}$ and Gumbel-Softmax with temperature τ and random samples $\mathbf{g}$.

In our implementation of VQ-VAE, we employed categorical reparameterization with Gumbel-Softmax [35] to achieve a simulated annealing effect as well as introduce stochasticity for variation in training:

$$\mathbf{q}_i = \sigma_{gsm}(\boldsymbol{\ell}_i, \tau) = \frac{exp(\frac{\boldsymbol{\ell}_i + \mathbf{g}_i}{\tau})}{\sum_{j=1}^{|\mathcal{C}|} exp(\frac{\boldsymbol{\ell}_j + \mathbf{g}_j}{\tau})} \tag{3.11}$$

where τ is a temperature parameter and σ_{gsm} is the Gumbel-Softmax function, which is an altered version of the softmax function with input $(\boldsymbol{\ell} + \mathbf{g})/\tau$. The elements of vector $\mathbf{g}$ are random samples from the Gumbel distribution

$$P_{Gumbel(\mu,\ \beta)}(x) = \frac{1}{\beta} e^{-(z+e^{-z})} \quad , \quad z = \frac{x-\mu}{\beta} \tag{3.12}$$

with $\mu = 0$ and $\beta = 1$. Starting from an initial temperature τ_0, the temperature is decreased by a multiplicative factor of α at each epoch k until reaching a final temperature τ_∞ such that

$$\tau_k = max\{\alpha^k \tau_0,\ \tau_\infty\}, \quad 0 < \alpha < 1, \quad 0 < \tau_\infty < \tau_0. \tag{3.13}$$

This cooling process makes the probability distribution steeper by each epoch to eventually become an argmax operation. This procedure aims to encourage the model to learn a discrete latent space with a small number of latent codes. A visualization of the bottleneck structure can be found in Fig. 3.4.

In order to level the selection of the codebook elements, VQ-VAE framework introduces an additional loss function, namely codebook loss. It uses the KL

divergence between the temperature-independent posterior categorical distributions and the uniform distribution ($\mathbf{u}$) such that the overall loss for VQ-VAE becomes

$$L^{\text{VQ-VAE}}_{\phi,\theta,\zeta} = L^{\text{AE}}_{\phi,\zeta,\theta} + \lambda_{kl} D_{KL}(\bar{\mathbf{q}}||\mathbf{u}) \tag{3.14}$$

where

$$\bar{\mathbf{q}} = \sigma_{max}(\boldsymbol{\ell}), \tag{3.15}$$

and λ_{kl} is the scale of the KL divergence from the uniform distribution. We select the scale λ_{kl} to be 0.0005 throughout the project.

3.1.2 Generative Adversarial Network

First introduced by Goodfellow et al. [17] in 2014, GANs are generative models that use two submodels, namely a generator (G_γ) and a discriminator (F_ψ), in a competitive setting to generate samples learned from an input distribution. Throughout the training, the discriminator is trained to distinguish between real and fake samples while the generator tries to deceive the discriminator with fake samples, using random samples from a predefined distribution as input. These two submodels are trained in an alternating fashion until the generator can generate samples that are indistinguishable from the real ones. The loss function for the overall model is

$$L^{\text{GAN}}_{\gamma,\psi}(x,\xi) = \mathbb{E}_{x\sim p_x}[F_\psi(x)] + \mathbb{E}_{\xi\sim p_\xi}[1 - F_\psi(\tilde{x})] \tag{3.16}$$

where $\tilde{x} = G_\gamma(\xi)$ is the output of the generator, $F_\psi(x)$ is the discriminator output, p_x and p_ξ are the data and random generator input distribution. The discriminator is trained to increase the loss while the generator is trained to decrease it such that

$$\gamma^*, \psi^* = \arg\min_{\gamma} \arg\max_{\psi} \; L^{\text{GAN}}_{\gamma,\psi}(x,\xi). \tag{3.17}$$

Considering the common problems in GAN training, especially vanishing gradients and mode collapse, we add a hinge gradient penalty term to the loss such that the overall loss becomes

$$L^{\text{WGAN}}_{\gamma,\psi}(x,z) = L^{\text{GAN}}_{\gamma,\psi}(x,\xi) + \lambda_{gp}\mathbb{E}_{\hat{x}\sim p_{\hat{x}}}[(\max\{0, \|\nabla_{\hat{x}} F_\psi(\hat{x})\|_2 - 1\})^2] \tag{3.18}$$

where λ_{gp} is set to 5 throughout our experiments and $\hat{x}$ is a random interpolation between input and the generated samples such that

$$\hat{x} = \eta x + (1-\eta)\tilde{x}, \; \eta \sim \text{Uniform}(0,1). \tag{3.19}$$

This version of GAN is known as a Wasserstein GAN (WGAN) [36] and the term introduced is Wasserstein gradient penalty [37]. The main advantage of this approach is the regularization of the gradient flow for the discriminator, which results in escaping from local minima and avoiding a vanishing gradient.

A visualization of the GAN can be found in Fig. 3.5.

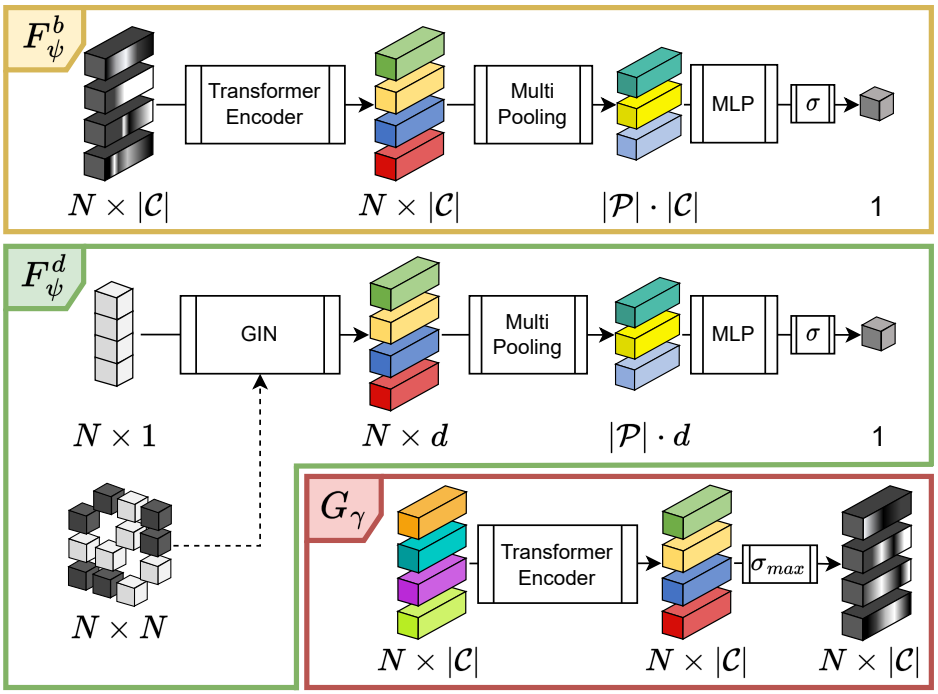


Figure 3.5: Overall GAN architecture.

Generator

For the generator part, we use a 2-layer transformer encoder with 8 heads, succeeded by a softmax layer along each node. The graph generator implementation has two different outputs: the codebook probabilities $\tilde{\mathbf{q}}$ and the mask for the nodes $\tilde{\mathbf{m}}$. The masks are generated statistically according to the distribution of the number of nodes N of the training set (see Fig. 4.1) and the last $N_{max} - N$ nodes are masked:

$$\tilde{\mathbf{m}} = [\texttt{ones}(N) \;||\; \texttt{zeros}(N_{max}(\mathbb{G}_{\text{TR}}) - N)], \quad N \sim p(N|\mathbb{G}_{\text{TR}}) \tag{3.20}$$

where we can derive the mask for the adjacency matrix as $\tilde{\mathbf{M}} = \tilde{\mathbf{m}}\tilde{\mathbf{m}}^\top$. The generator takes an input noise sampled from a Gaussian distribution with mean 0 and variance 1.

Discriminator

For the discriminator part, we construct two different versions: namely bottleneck and decoder discriminators. The bottleneck discriminator aims to discriminate the posterior categorical distributions (F_ψ^{b}) while the decoder discriminator differentiates the adjacency matrices (F_ψ^{d}). Both discriminators are permutation

invariant on graph nodes. This is achieved by using multiple pooling functions ($\mathcal{P}$) on nodes, which is inspired by the pooling mechanism in principal neighbourhood aggregation by Corso et al. [38]. We use four types of pooling functions, namely `maximum`, `minimum`, `summation`, and `standard deviation`. The pooling is applied over only the non-masked nodes.

3.1.3 Module Details

Transformer Encoder and Attention Mechanism

The rationale behind the attention is to introduce a mechanism which defines the importance of the information for a given task. In this sense, the attention function highlights or downplays some characteristics of an input value ($\mathbf{v}$) according to the cosine similarity of a query ($\mathbf{q}$) and a key ($\mathbf{k}$) set:

$$\mathbf{y} = \text{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \sigma_{max}\left(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{d_k}}\right)\mathbf{v} \tag{3.21}$$

where the key vector is divided by the square root of the dimension in order to normalize its variance. Although there are various methods, in its most known version of self-attention, these three inputs are computed from the input ($\mathbf{x}$) using linear layers:

$$\mathbf{q} = W_q^\top \mathbf{x}, \quad \mathbf{k} = W_k^\top \mathbf{x}, \quad \mathbf{v} = W_v^\top \mathbf{x}. \tag{3.22}$$

Furthermore, we can use numerous self-attention units simultaneously with different query $\mathbf{q}_i$, key $\mathbf{k}_i$, and value $\mathbf{v}_i$ vectors owing to the parallel computational power of GPUs. Next, a linear layer outputs the concatenated results of these outputs.

$$\mathbf{y} = W_o^\top [\mathbf{y}_1 \,||\, \mathbf{y}_2 \,||\, ... \,||\, \mathbf{y}_h]. \tag{3.23}$$

This parallelization of the self-attention mechanism is referred to as multi-head attention (MHA). This method is beneficial because it extracts higher-level features by focusing multiple attentions on different input data structures.

As explained in [39], a transformer encoder layer is a structure of residual multi-head attentions followed by residual feedforward networks (FFNs), with layer normalization (LN) in between. Thus, the output of the i^{th} encoder layer can be represented mathematically as:

$$\begin{aligned} \mathbf{y}_i =& \text{LN}(\mathbf{z}_{i-1} + \text{MHA}(W_q^\top \mathbf{z}_{i-1}, W_k^\top \mathbf{z}_{i-1}, W_v^\top \mathbf{z}_{i-1})), \\ \mathbf{z}_i =& \text{LN}(W_2^\top (W_1^\top \mathbf{y}_i + b_1) + b_2 + \mathbf{y}_i). \end{aligned} \tag{3.24}$$

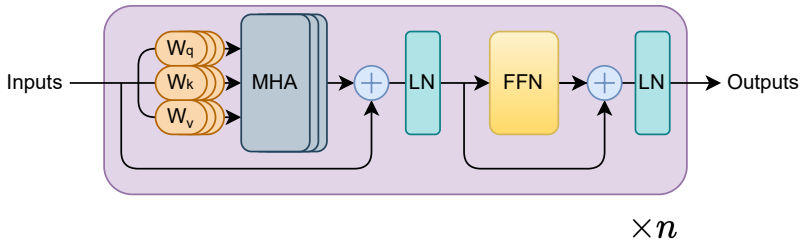


Figure 3.6: Transformer encoder structure which consists of n consecutive self-attention blocks (SAB).

3.2 Training Procedure

For the training of the overall model, we follow a greedy approach: First, we train the VQ-VAE part of the model and then we teach the GAN part. For the configuration with the bottleneck discriminator (F_ψ^{b}), the autoencoder weights are frozen while the GAN weights are trained. For the configuration with the decoder discriminator (F_ψ^{d}), only the decoder weights of the autoencoder are trained together with the GAN.

3.2.1 Autoencoder

During the training of the autoencoder, we use the loss described in Eq. 3.14. After a series of hyperparameter searches, we decide on two main settings for the autoencoder training, namely validation (VL) and training (TR). The former is our standard configuration, which uses the normalized binary cross entropy (Eq. 3.8) for the vanilla part of the VQ-VAE loss and considers the highest F_1 score in the validation set for early stopping. In contrast, the latter is mainly used to understand the possible outcomes of overfitting for the GAN training. In this sense, in the TR configuration, we use the training set for validation and the standard binary cross entropy (Eq. 3.7) for the vanilla part of the overall loss, which is not regularized compared to the normalized version. In our experiments, we see that the normalized binary cross entropy is better for both edge and no-edge precision while the standard one achieves better recall values and mostly better F_1 score. The evaluation of the autoencoder is done on a test set using the autoencoder weights in the epoch with the best F_1 score for validation and training sets in the VL and TR configurations, respectively. A detailed description of the metrics used in autoencoder training can be found in Section 4.1.1.

For both configurations, we use an AdamW optimizer [40] with a learning rate of 0.0003, betas $\{0.5, 0.9\}$, and weight decay of 0.1. We train the autoencoder for 30 000 epochs with a batch size of 20. We use a cosine warm-up scheduler with

3 000 epochs of warm-up duration. For the Gumbel-Softmax, we select starting and final temperature pair of $\{2.0, 0.2\}$ with a cooling rate of 0.9998. A detailed set of hyperparameters for the VQ-VAE can be found in Table B.1.

3.2.2 Generative Adversarial Network

As described in Sections 3.2.1 and 3.1.2, there are already two configuration decisions with two options: the type of the autoencoder training process and the placement of the discriminator. With the two additional options for the temperature selection of Gumbel-Softmax, overall 8 different configurations are considered for the GAN training:

1. **Autoencoder type**: Validation (VL) vs. Training (TR)
2. **Discriminator type**:
   Discriminator at Bottleneck (DAB) vs. Discriminator at Decoder (DAD)
3. **Temperature selection**:
   Constant (CST) vs. Desired Mean of Max (DMM)

Algorithm 1 Temperature for Desired Mean of Max

Require: training set $\mathbb{G}_{\text{TR}}$, encoder E_ϕ, tolerance ϵ, desired mean of max $\hat{m}$
Ensure: temperature τ

Calculate logits for the training set: $\mathbf{L} \leftarrow E_\phi(\mathbb{G}_{\text{TR}})$
Set initial temperature: $\tau \leftarrow \tau_0$
while True do
  Computing posterior categorical distribution with logits over temperature: $\tilde{\mathbf{Q}} \leftarrow \sigma_{max}(\mathbf{L}/\tau)$
  Taking max of posterior categorical distribution: $\mathbf{m} \leftarrow \max_i \tilde{\mathbf{Q}}_i$
  Taking mean of max of posterior categorical distribution: $m \leftarrow |\mathbb{V}(\mathbb{G}_{\text{TR}})|^{-1} \cdot \sum_j \mathbf{m}_j$
  Calculating the difference with desired: $\Delta \leftarrow m - \hat{m}$
  if $|\Delta| < \epsilon$ **then**
    break
  end if
  Update temperature: $\tau \leftarrow \tau \cdot exp(\Delta)$
end while
return τ

In the constant temperature configuration, we use the default temperature of 1.0 throughout the training while we set the temperature to achieve a DMM of posterior categorical distributions of 0.2 on the training set, where we select the value after a grid search of possible DMMs of $\{0.2, 0.5, 0.8\}$. For a set of graphs

with nodes $\mathbb{V} = \mathcal{V}_{i=1}^{|\mathbb{G}|} = \{v_i\}_{i=1}^{|\mathbb{V}|}$, we can define the mean of max of posterior categorical distributions as:

$$\texttt{mean\_of\_max}(\tilde{\mathbf{q}}) = \frac{1}{|\mathbb{V}|} \sum_{j=1}^{|\mathbb{V}|} \max_{i=1:|\mathcal{C}|} \{\tilde{\mathbf{q}}_i\}_j \tag{3.25}$$

where the Gumbel random vector has been omitted in the calculation of posterior categorical distributions for statistical stability

$$\tilde{\mathbf{q}}_i = \sigma_{max}(\frac{\boldsymbol{\ell} + \mathbb{E}[\mathbf{g}]}{\tau}) = \sigma_{max}(\frac{\boldsymbol{\ell} + \mathbf{0}}{\tau}) = \sigma_{max}(\frac{\boldsymbol{\ell}}{\tau}). \tag{3.26}$$

Overall, posterior categorical distributions not only have the information of the node embeddings but also represent the relationship between codebook elements. In the extreme case of very low temperature, where the mean of max is close to 1, the information on codebook characteristics becomes intractable by the GAN. In the other extreme case of very hot temperature, where all the distributions become almost uniform, no information remains at all. The rationale behind the use of DMM is that defining the mean of max directly to avoid both extrema is a more convenient approach instead of defining a random temperature and hoping it is not too cold or hot.

We use an iterative process to achieve this temperature with a tolerance value of 0.001. The algorithm for the DMM is described in Alg. 1.

For the GAN, we use the loss described in Eq. 3.18. The optimization is done with an AdamW optimizer [40] with a learning rate of 0.0001, betas $\{0.5, 0.9\}$, and weight decay of 0.0001. We train the GAN for 10 000 epochs with a batch size of 20. A detailed set of hyperparameters of the model can be found in Table B.2.

Chapter 4

Evaluation

4.1 Metrics

4.1.1 Reconstruction

For the evaluation of the reconstructed graphs, we consider the error on each non-diagonal element of the reconstructed hard adjacency matrix, where the elements of the adjacency matrix are forced to be 1 or 0 according to their closeness, using the following metrics:

- **Precision**: We use two types of precisions, one for the edge elements and one for the no-edge elements. They are defined as TP/(TP + FP) and TN/(TN + FN), respectively.
- **Recall**: For the recall, we consider only the edge elements for the sake of simplicity. We use the following definition: TP/(TP + FN).
- **F_1 Score**: We use the following standard definition:
  $2 \times \text{Precision(Edge)} \times \text{Recall}/(\text{Precision(Edge)} + \text{Recall})$.

where TP, TN, FP, and FN are the number of true positives, true negatives, false positives, and false negatives, respectively.

4.1.2 Generation

For the evaluation of generated graphs, we use Maximum Mean Discrepancy (MMD) which is also used by Martinkus et al. [41] on three different graph characteristics. The MMD is a distance measured between two distributions and is defined as

$$\text{MMD}_k(P, Q) := \|\mu_P - \mu_Q\|_{\mathcal{H}} \quad , \quad \mu_P := \int k(\cdot, x) dP(x) \tag{4.1}$$

where $\mathcal{H}$ is a Reproducing Kernel Hilbert Space (RKHS) [42]. We used a Gaussian total variation (TV) kernel with a variance (σ^2) of 1:

$$k_{\mathcal{H}}(X,Y) = exp\left(\frac{1}{2\sigma}\left(\sum_i \frac{|p_i(X) - p_i(Y)|}{2}\right)^2\right). \tag{4.2}$$

where $p_i(X)$ is the i^{th} index of the discrete probability distribution. The graph characteristics used for the MMD are:

- Node Degree **d**:
$$d(v_i) = \sum_j \mathbf{A}_{i,j} \tag{4.3}$$
which is the number of edges incident to node v_i.
- Local Clustering Coefficient **c**:
$$c(v_i) = \frac{2T(v_i)}{d(v_i)(d(v_i)-1)} \tag{4.4}$$
where $T(v_i)$ is the number of triangles including node v_i.
- Laplacian Spectra $\boldsymbol{\lambda}(\bar{\mathcal{L}})$: Eigenvalues of normalized Laplacian matrix ($\bar{\mathcal{L}}$)
$$\bar{\mathcal{L}} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-1/2} \tag{4.5}$$
where $\mathbf{D}$ is the diagonal matrix of node degrees ($\mathbf{d}$).

Other than these three, we look for the ratio of the generated samples that are nonisomorphic to all graphs in the compared set, which we defined as **novelty** measure. Moreover, we check the ratio of how many graphs are considered as **valid** via the specific algorithms for the graph type.

As a general measure of success, we use the ratios of MMDs where the nominator is the MMD between generated graph and the set of ground truth graphs while the denominator is the MMD between the training set and the ground truth set. We take the average of the ratios

$$R_{\text{MMD}}(\tilde{\mathbb{G}}||\mathbb{G}_{\mathcal{S}}) = \frac{1}{3}\sum_{\mathcal{M}\in\{\mathbf{d},\mathbf{c},\boldsymbol{\lambda}\}} \frac{\text{MMD}^{\mathcal{M}}(\tilde{\mathbb{G}},\mathbb{G}_{\mathcal{S}})}{\text{MMD}^{\mathcal{M}}(\mathbb{G}_{\text{TR}},\mathbb{G}_{\mathcal{S}}) + \epsilon} \tag{4.6}$$

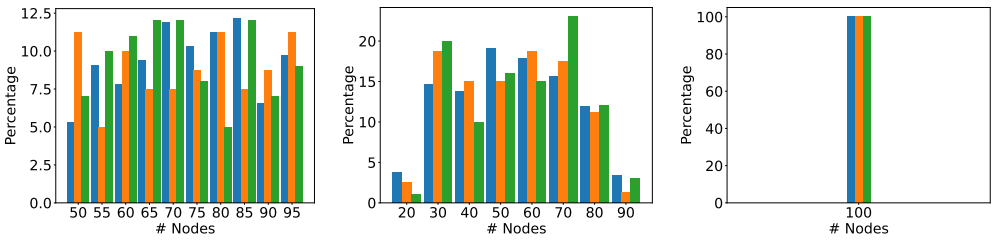
where ϵ is a small constant (10^{-5}) which is added to the denominator to avoid division by zero.

4.2 Datasets

Throughout our work, we use three procedurally generated datasets consisting of 500 nonisomorphic graphs. All datasets are separated into training, validation, and test sets including 320, 80, and 100 samples, respectively.

- **Tree** is any acyclic graph without a cycle. We generate tree graphs with 100 nodes.
- **Lobster** is any tree graph where all nodes are within a distance of 2 from the defined central path, which is also known as the backbone. We generate lobsters with an average number of 75 nodes. The minimum number of nodes is 50 and the maximum number of nodes is 100. We use 0.7 for both the probability of adding an edge to the backbone and the probability of adding an edge one level beyond the backbone.
- **Stochastic Block Model** (SBM) graphs are random graphs with a tendency to have many small densely connected components, which are also known as communities. We build SBMs randomly with $[2, 5]$ communities and $[10, 20]$ nodes, which effectively gives the maximum number of nodes 100. The inter-community edge probability is 0.3 and the intra-community edge probability is 0.05.

All the graphs used in the datasets are generated procedurally using the Python NetworkX [43] library. The distribution of the number of nodes in the graph datasets can be found in Fig. 4.1.



(a) Lobster (b) SBM (c) Tree

Figure 4.1: Normalized histograms of the number of nodes in the graph datasets. The blue, orange, and green bars show the number of nodes in the training, validation, and test sets, respectively.

Dataset	Comparison	Degree	Clustering	Spectral
Lobster	TR-VL	1.372E-04	0.000E+00	7.601E-04
Lobster	TR-TE	2.845E-04	0.000E+00	5.944E-04
Lobster	VL-TE	7.183E-05	0.000E+00	8.967E-04
SBM	TR-VL	2.886E-04	1.537E-02	2.978E-03
SBM	TR-TE	1.181E-04	1.358E-02	3.690E-03
SBM	VL-TE	6.918E-04	2.160E-02	6.572E-03
Tree	TR-VL	6.667E-05	0.000E+00	3.457E-03
Tree	TR-TE	4.298E-05	0.000E+00	1.681E-03
Tree	VL-TE	7.027E-05	0.000E+00	4.527E-03

Table 4.1: MMD results between the sets of graph datasets. TR, VL, and TE are abbreviations for training, validation, and test sets.

4.3 Results

4.3.1 Reconstruction

For the reconstruction part of the project, we train the VQ-VAE in two different configurations: training (TR) and validation (VL), as described in Section 3.2.1 in detail. Considering the results showed in Table 4.2, 4.3, and 4.4, the former configuration is more successful in recall and mostly in F_1 score while the latter is better at precision. Regarding the datasets, we can order the datasets from easiest to most difficult to construct as lobster, tree, and SBM. In conclusion, because the errors in the reconstruction could be overridden or used as a randomness source in the generation part and the general structure of the graphs is somewhat preserved, especially in the TR configuration, we can say that the graph reconstruction might be enough for our requirements but it is not satisfactory in general.

The values of reconstruction metrics during training is given in Fig. 4.2, A.1, A.2, and A.3. Moreover, a visualization of randomly selected samples can be found in Fig. 4.3, A.4, and A.5.

Set Type	Configuration	F_1 Score	Recall	% Precision
Training Set	Training	0.923	0.927	91.9-99.8
Training Set	Validation	0.705	0.545	99.7-97.9
Validation Set	Training	0.769	0.740	80.0-99.3
Validation Set	Validation	0.701	0.541	99.6-97.9
Test Set	Training	**0.775**	0.755	79.5-99.3
Test Set	Validation	0.692	0.531	99.5-97.7

Table 4.2: Results of the graph reconstruction experiments using the VQ-VAE on the lobster dataset.

Set Type	Configuration	F_1 Score	Recall	% Precision
Training Set	Training	0.919	0.926	91.3-99.4
Training Set	Validation	0.495	0.331	98.1-82.8
Validation Set	Training	0.439	0.443	43.4-95.2
Validation Set	Validation	0.495	0.332	98.0-82.6
Test Set	Training	0.423	0.427	42.0-95.3
Test Set	Validation	**0.475**	0.314	97.9-82.0

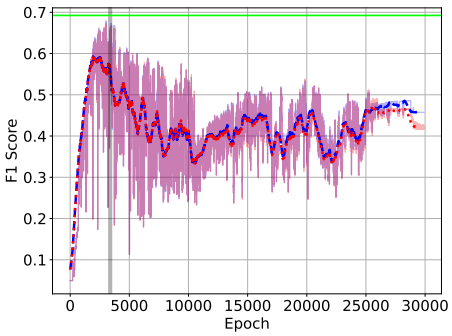
Table 4.3: Results of the graph reconstruction experiments using the VQ-VAE on the SBM dataset.

Set Type	Configuration	F_1 Score	Recall	% Precision
Training Set	Training	0.765	0.801	73.3-99.6
Training Set	Validation	0.586	0.424	95.1-97.4
Validation Set	Training	0.652	0.659	64.4-99.3
Validation Set	Validation	0.558	0.396	94.3-97.1
Test Set	Training	**0.665**	0.669	66.2-99.3
Test Set	Validation	0.564	0.403	94.2-97.2

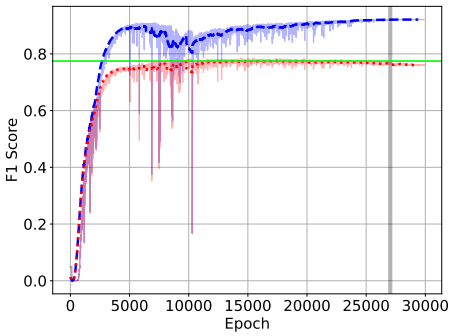
Table 4.4: Results of the graph reconstruction experiments using the VQ-VAE on the tree dataset.

4.3.2 Generation

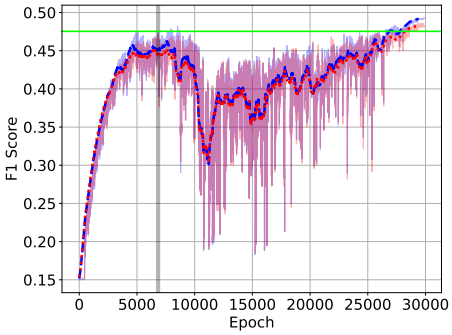
For the graph generation, we use 8 different configurations per dataset, regarding the training type of the VQ-VAE, discriminator type, and temperature selection during GAN training. The details of these configurations are previously mentioned in Section 3.2.2. Considering the results showed in Table 4.5, 4.6, and 4.7, the average MMD ratios calculated according to Eq. 4.6 suggest that the generated graphs are not successful in expression of the ground truth graph structures. Considering the validity of graphs, our model is better at preserving the general structure of the lobster and tree graphs while this is not the case for SBM. However, it should be noted that the generated graphs for lobster and tree graphs have very few nodes since the isolated nodes are removed at the end of the generation process. This increases the chance of a random graph being considered a valid one. In addition to these, contrary to the success of the TR configuration in reconstruction, the lowest MMD ratio is always achieved with the VL configuration for test sets. This is most possibly due to the non-overfitting behaviour of the VQ-VAE in the VL setting which allows a more generalizable understanding of the graph type to generate. There is no significant relationship between the success of the model and the selection of temperature. We can make a similar observation for the discriminator type. Finally, as a side product of the inexpressiveness of the overall framework, the ratio of the novel graphs is 1.0 in



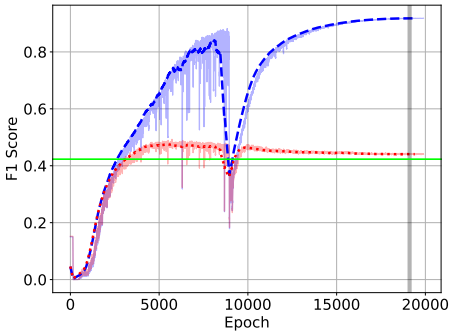
(a) Lobster in Validation Configuration



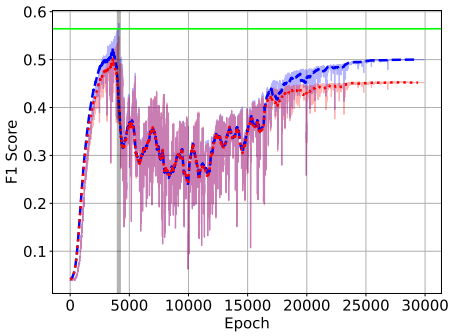
(b) Lobster in Training Configuration



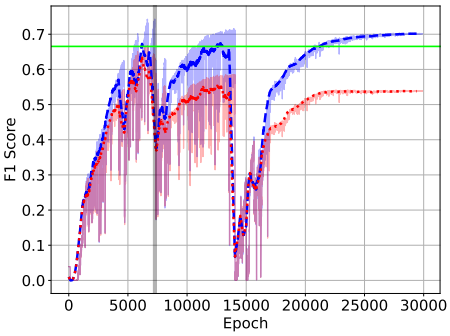
(c) SBM in Validation Configuration



(d) SBM in Training Configuration



(e) Tree in Validation Configuration

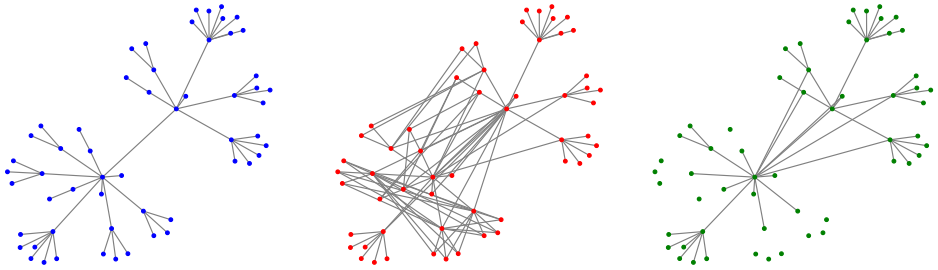


(f) Tree in Training Configuration

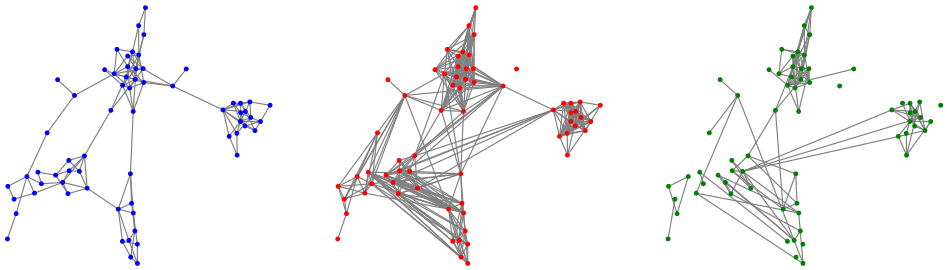
Figure 4.2: Epochs vs. F_1 Scores achieved by the VQ-VAE for different configurations and datasets. Validation, training, test results, and best epoch are shown in colours blue, red, green, and grey, respectively.

all of the experiments.

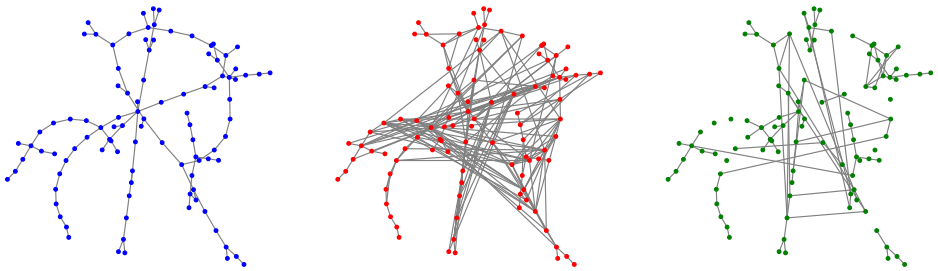
Other than the numeric results of the graph generation, we can also make a visual inspection of the generated graphs showed in Fig. A.6, A.7, and A.8. It is evident that the generated graphs are not similar to the ground truth graph



(a) Sample from the lobster test set



(b) Sample from the SBM test set



(c) Sample from the tree test set

Figure 4.3: Reconstructions of the test graphs by the VQ-VAE. From left to right: ground truth (blue), reconstruction in VL (red), reconstruction in TR (green).

structures for all three datasets. In contrast to our findings on numeric results, generated graphs for SBM seem to have relatively better similarity to the ground truth graph structure. It can be seen that there is a certain effort to construct communities even though there is no connection between them. On the contrary, both generated graphs for lobster and tree show a central structure as one or a few nodes.

AE	Disc.	Temp.	Degree	Clustering	Spectral	Ratio	Valid	Novelty
TR	DAB	CST	6.75E-02	4.33E-03	1.39E-01	2.97E+02	**0.30**	1.00
TR	DAB	DMM	8.57E-02	2.03E-02	1.39E-01	8.52E+02	0.24	1.00
TR	DAD	CST	8.84E-02	5.90E-12	6.58E-01	4.63E+02	0.07	1.00
TR	DAD	DMM	3.00E+02	5.90E-07	1.13E+03	9.66E+05	0.06	1.00
VL	DAB	CST	4.93E-02	2.01E-12	2.43E-01	**1.90E+02**	0.06	1.00
VL	DAB	DMM	1.28E-01	5.56E-02	1.51E-01	2.08E+03	0.03	1.00
VL	DAD	CST	4.65E-02	3.41E-12	3.78E-01	2.61E+02	0.04	1.00
VL	DAD	DMM	2.21E+02	2.00E+03	9.96E+02	6.75E+07	0.09	1.00

Table 4.5: Results of graph generation on the test set of the lobster dataset.

AE	Disc.	Temp.	Degree	Clustering	Spectral	Ratio	Valid	Novelty
TR	DAB	CST	3.63E+02	4.15E+00	8.27E+01	9.51E+05	0.00	1.00
TR	DAB	DMM	4.31E+02	5.03E+00	6.86E+01	1.13E+06	0.00	1.00
TR	DAD	CST	2.44E+03	1.98E+00	3.33E+01	6.35E+06	0.00	1.00
TR	DAD	DMM	8.70E+02	1.98E+00	1.59E+01	2.26E+06	0.00	1.00
VL	DAB	CST	5.49E+02	7.62E+00	5.97E+01	1.43E+06	0.00	1.00
VL	DAB	DMM	6.70E+02	2.12E+00	4.75E+01	1.75E+06	0.00	1.00
VL	DAD	CST	1.22E+03	1.97E+00	1.87E+01	3.17E+06	0.00	1.00
VL	DAD	DMM	1.14E-01	3.30E-02	1.64E-01	**3.11E+02**	0.00	1.00

Table 4.6: Results of graph generation on the test set of the SBM dataset.

AE	Disc.	Temp.	Degree	Clustering	Spectral	Ratio	Valid	Novelty
TR	DAB	CST	9.54E-02	7.17E-03	4.02E-01	9.19E+02	**0.42**	1.00
TR	DAB	DMM	5.78E-02	1.85E-02	3.90E-01	1.06E+03	0.14	1.00
TR	DAD	CST	2.29E-01	1.01E+00	1.82E-01	3.52E+04	0.00	1.00
TR	DAD	DMM	5.01E+03	5.02E-07	3.10E+02	3.16E+07	0.04	1.00
VL	DAB	CST	1.29E-01	2.42E-04	4.30E-01	9.03E+02	0.11	1.00
VL	DAB	DMM	9.29E-02	6.11E-03	3.91E-01	**8.66E+02**	0.19	1.00
VL	DAD	CST	4.85E+03	4.08E+03	3.20E+02	1.67E+08	0.06	1.00
VL	DAD	DMM	2.46E-01	7.81E-03	5.30E-01	1.91E+03	0.14	1.00

Table 4.7: Results of graph generation on the test set of the tree dataset.

Chapter 5

Discussion and Future Work

5.1 Discussion

Regarding the evaluation of our framework, we are aware of various problems and weak points:

- **Insufficiency of VQ-VAE**: The results of the VQ-VAE are mostly insufficient and the main training setting, namely VL, seems over-regularized. In this manner, selecting a more stable training procedure would have been beneficial. Moreover, the expressive power of the GNNs in the VQ-VAE is questionable, regarding the number of samples required to have a better validation loss is relatively high compared to the other graph autoencoder models. One cause for this is the imposed permutation equivariance, which limits the flexibility, and eventually generalizability of the model to some extent. Overall, the insufficiency of VQ-VAE results in an error accumulation from the graph reconstruction process to the graph generation.
- **VQ-VAE as an instability source for GAN:** Training a GAN already has difficulties due to the instability brought by the competitiveness between the generator and the discriminator. Although we use a Wasserstein gradient penalty term to avoid such problems, it is possible that the bias introduced by the VQ-VAE also becomes an instability source. As an argument in favour of this observation, we can show the successful results of our GAN on Gaussian distribution generation described in Section C.1.
- **Independent mask generation:** The mask generation is independent of the graph generation, which means that the mask generation is not aware of the graph structure. This might result in unwanted artefacts in the generated graphs.
- **Complexity of one-shot generation:** Compared to the autoregressive approaches, one-shot methods require constructing a new sample from scratch. This is a relatively more difficult task for the generator, especially in the use of a permutation equivariant model, where the relationship

between the vertices should be learned only by the response of the discriminator. In general, this requires longer training with more data and most possibly a more expressive generator architecture.

- **Lack of a thorough hyperparameter search:** Due to the period of the project and the high computational requirements of the framework, we could not perform a thorough hyperparameter search. Instead, the search is mostly done in a greedy fashion during the development of the modules, along with the evaluation of the module themselves. Hyperparameter search is especially crucial for the GAN, for which the training process is very sensitive to hyperparameters.

5.2 Future Work

Considering the problems mentioned in the previous section, we discuss the possible improvements for future work:

- Regarding the error accumulation from the graph reconstruction process to the graph generation, the VQ-VAE should be improved. One of the problems with the VQ-VAE is the training procedure, considering the over-regularization and over-fitting behaviour in TR and VL configurations, respectively. Instead, a midpoint training configuration can be used to avoid both extrema, such as using the standard binary cross entropy loss in the VL configuration. Still, the results suggest that changing the training configurations only most possibly would not be enough to achieve satisfactory results in reconstruction. In this sense, an overall inspection of the VQ-VAE architecture is also necessary.

- A possible, direct and easy-to-implement improvement on the VQ-VAE is using more informative initial GIN vectors instead of the node mask vector. These vectors might contain information such as the degree or spectral properties of the nodes, which might allow the VQ-VAE to capture the node properties better and eventually increase the overall expressiveness of the model. Moreover, in a later stage, other permutation equivariant GNNs can be tried both for the encoder and decoder parts. In the final case, the overall framework should be rethought. In this sense, a possible means of improvement can be introducing an independent decoder to be used in GAN to compensate for the imperfect decodings of the VQ-VAE.

- The transformer encoder used as the generator can be changed with a transformer encoder-decoder. Similar to the Taming Transformers [31] and GraphRNN [2], such an autoregressive generator could capture the relationship between the generated nodes better compared to the one-shot approach. Moreover, the mask generation process could be integrated into the

graph generation process using an end token. Even though the generation procedure itself is not permutation equivariant, thanks to the permutation equivariance of the decoder, such structural change will not induce a loss in the overall permutation equivariance of the framework [44]. The main disadvantage of using an autoregressive approach for generation is the increase in the training and inference durations, which results in a compromise in scalability, eventually.

- Increasing the size of the datasets is a natural solution to increase the success of VQ-VAE and alleviate the instability introduced during GAN training and overall error accumulation. A solution without changing the total number of samples is training the VQ-VAE on a combined dataset containing multiple graph types while training a different GAN for each type after.

- To inspect the individual roles of the VQ-VAE and the GAN, we can train the GAN on a different dataset from what the VQ-VAE is trained with. This would allow us to see whether the relatively good results we see in Fig. A.7 are the success of the GAN or the inherent bias of the VQ-VAE to reconstruct the graphs. In addition, for the assessment of the inter-node perception of the GAN architecture independent from the VQ-VAE, we can conduct an additional study on two-dimensional distribution generation, where the new dimension is defined as the node space.

- The hyperparameters of the overall framework can be tuned more thoroughly. This would allow us to find a better balance between the submodels, especially between the generator and the discriminator.

5.3 Conclusion

In conclusion, we present a novel permutation equivariant framework to generate synthetic graphs using a VQ-VAE with Gumbel-Softmax and a GAN with the Wasserstein gradient penalty. The training procedure of our model is done greedily by training the VQ-VAE first and using the learned posterior categorical distributions for vertices as embedding space during the training of the GAN. For evaluation, we use graph datasets that are procedurally generated as three well-known graph types: lobster, SBM, and tree. For the VQ-VAE part, we obtain unsatisfactory results on graph reconstruction. Similar to the VQ-VAE, the GAN part is not able to generate realistic graphs and the final results are far from perfect. Still, the relatively good results we see in Fig. A.7 show that the model might have some potential in the end. To fulfil this potential, the improvements to the framework should be done in accordance with a careful inspection of the individual modules it consists of.

Bibliography

[1] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.

[2] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *International conference on machine learning*. PMLR, 2018, pp. 5708–5717.

[3] Q. Liu, M. Allamanis, M. Brockschmidt, and A. Gaunt, "Constrained graph variational autoencoders for molecule design," *Advances in neural information processing systems*, vol. 31, 2018.

[4] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[5] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[6] I. Krawczuk, P. Abranches, A. Loukas, and V. Cevher, "GG-GAN: A geometric graph generative adversarial network, 2021," in *URL https://openreview.net/forum*, 2021.

[7] E. N. Gilbert, "Random graphs," *The Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, 1959.

[8] P. Erdos, A. Rényi *et al.*, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[9] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: an approach to modeling networks." *Journal of Machine Learning Research*, vol. 11, no. 2, 2010.

[10] S. Wasserman and P. Pattison, "Logit models and logistic regressions for social networks: I. an introduction to markov graphs andp," *Psychometrika*, vol. 61, no. 3, pp. 401–425, 1996.

[11] G. Bianconi and A.-L. Barabási, "Bose-Einstein condensation in complex networks," *Physical review letters*, vol. 86, no. 24, p. 5632, 2001.

[12] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE access*, vol. 7, pp. 53 040–53 065, 2019.

[13] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[14] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[15] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[16] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[18] D. Saxena and J. Cao, "Generative adversarial networks (GANs) challenges, solutions, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–42, 2021.

[19] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[20] M. Simonovsky and N. Komodakis, "GraphVAE: Towards generation of small graphs using variational autoencoders," in *International conference on artificial neural networks*. Springer, 2018, pp. 412–422.

[21] J. Mitton, H. M. Senn, K. Wynne, and R. Murray-Smith, "A graph VAE and graph transformer approach to generating molecular graphs," *arXiv preprint arXiv:2104.04345*, 2021.

[22] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng, "Recent progress on generative adversarial networks (GANs): A survey," *IEEE Access*, vol. 7, pp. 36 322–36 333, 2019.

[23] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, W. Li, X. Xie, and M. Guo, "Learning graph representation with generative adversarial nets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 8, pp. 3090–3103, 2019.

[24] N. De Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," *arXiv preprint arXiv:1805.11973*, 2018.

[25] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *International conference on machine learning*. PMLR, 2018, pp. 610–619.

[26] C. Yang, P. Zhuang, W. Shi, A. Luu, and P. Li, "Conditional structure generation through graph variational generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[27] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, "Efficient graph generation with graph recurrent attention networks," *Advances in neural information processing systems*, vol. 32, 2019.

[28] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky, "Graph normalizing flows," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[29] C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon, "Permutation invariant graph generation via score-based generative modeling," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 4474–4484.

[30] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[31] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 873–12 883.

[32] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[33] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[34] H. Serviansky, N. Segol, J. Shlomi, K. Cranmer, E. Gross, H. Maron, and Y. Lipman, "Set2graph: Learning graphs from sets," *Advances in Neural Information Processing Systems*, vol. 33, pp. 22 080–22 091, 2020.

[35] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with Gumbel-Softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[36] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

[37] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," *Advances in neural information processing systems*, vol. 30, 2017.

[38] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 260–13 271, 2020.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[40] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2018.

[41] K. Martinkus, A. Loukas, N. Perraudin, and R. Wattenhofer, "SPECTRE: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators," *arXiv preprint arXiv:2204.01613*, 2022.

[42] I. O. Tolstikhin, B. K. Sriperumbudur, and B. Schölkopf, "Minimax estimation of maximum mean discrepancy with radial kernels," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[43] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using NetworkX," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[44] C. Vignac and P. Frossard, "Top-N: Equivariant set and graph generation without exchangeability," *arXiv preprint arXiv:2110.02096*, 2021.

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[46] NVIDIA, P. Vingelmann, and F. H. Fitzek, "CUDA, release: 11.3," 2022. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[47] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," *arXiv preprint arXiv:1903.02428*, 2019.

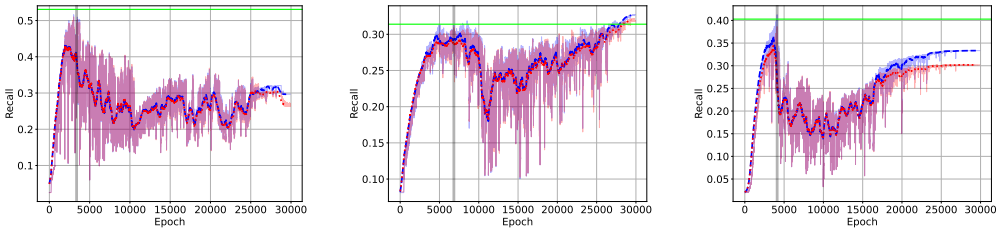
[48] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

Appendix A

Additional Results

A.1 Reconstruction

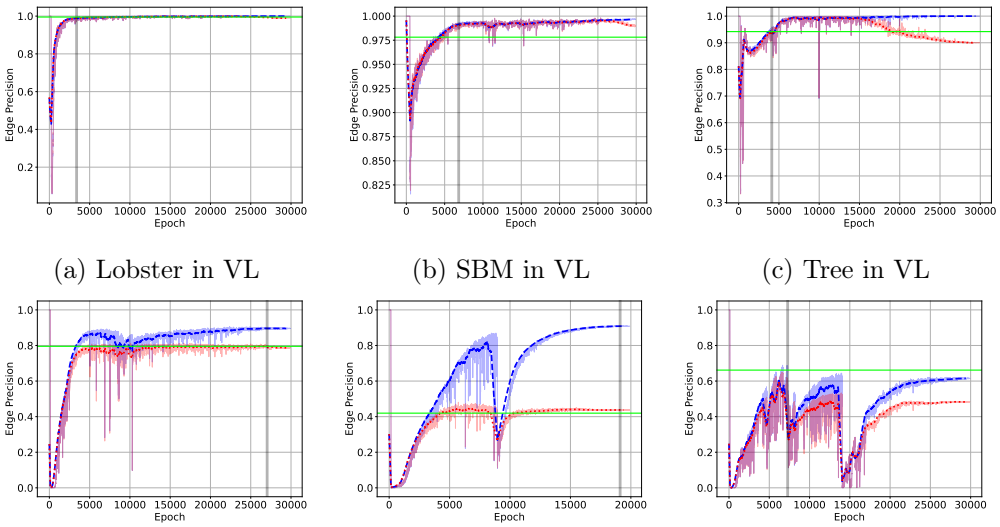
A.1.1 Epochs



(a) Lobster in VL (b) SBM in VL (c) Tree in VL

(d) Lobster in TR (e) SBM in TR (f) Tree in TR

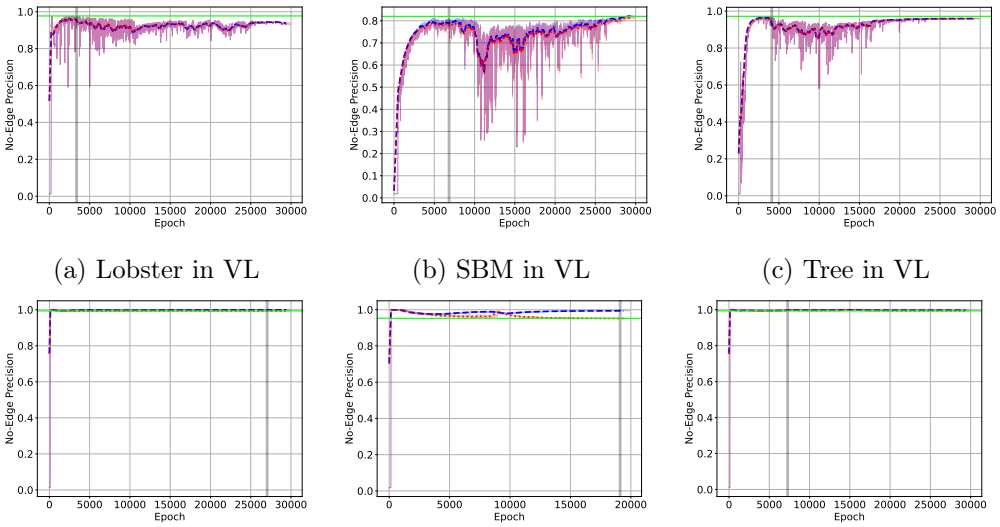
Figure A.1: Epochs vs. Recalls achieved by the VQ-VAE for different configurations and datasets. Validation, training, test results, and best epoch are shown in colours blue, red, green, and grey, respectively.



(a) Lobster in VL (b) SBM in VL (c) Tree in VL

(d) Lobster in TR (e) SBM in TR (f) Tree in TR

Figure A.2: Epochs vs. Edge Precisions achieved by the VQ-VAE for different configurations and datasets. Validation, training, test results, and best epoch are shown in colours blue, red, green, and grey, respectively.

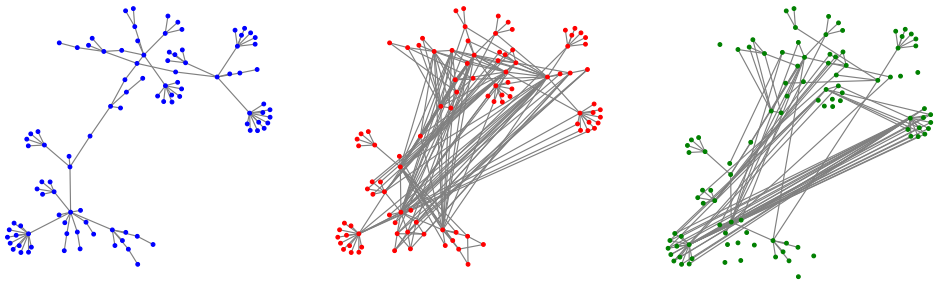


(a) Lobster in VL (b) SBM in VL (c) Tree in VL

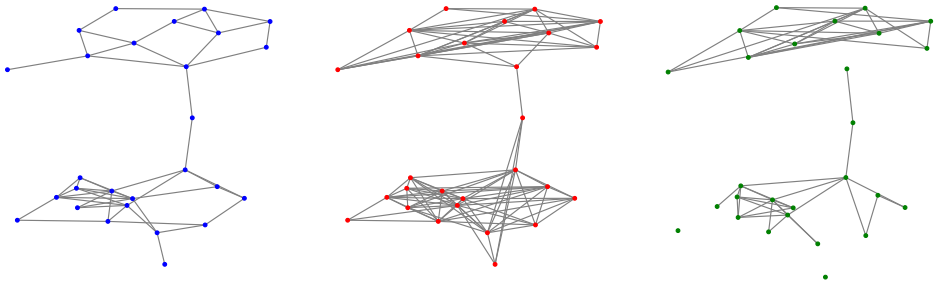
(d) Lobster in TR (e) SBM in TR (f) Tree in TR

Figure A.3: Epochs vs. No-Edge Precision achieved by the VQ-VAE for different configurations and datasets. Validation, training, test results, and best epoch are shown in colours blue, red, green, and grey, respectively.

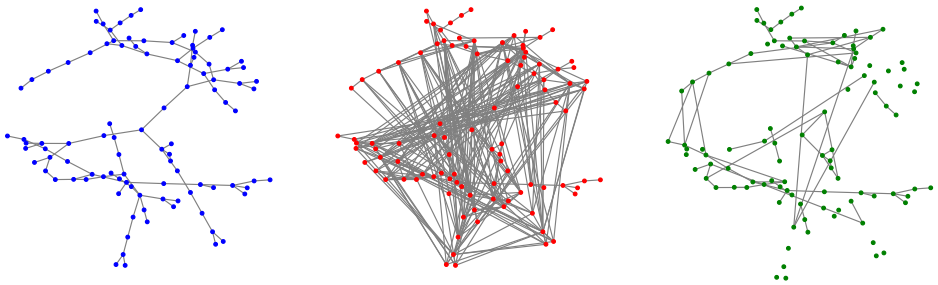
A.1.2 Visualization



(a) Sample from the lobster validation set

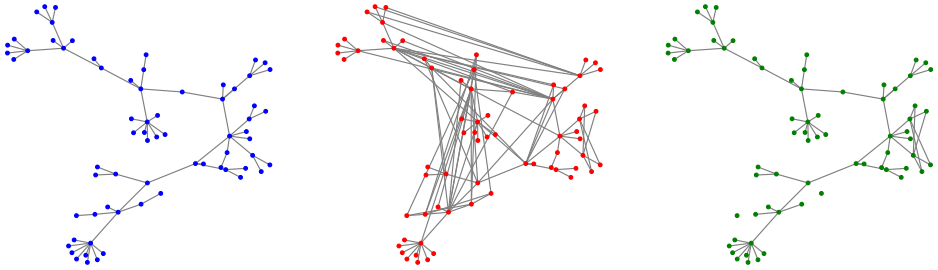


(b) Sample from the SBM validation set

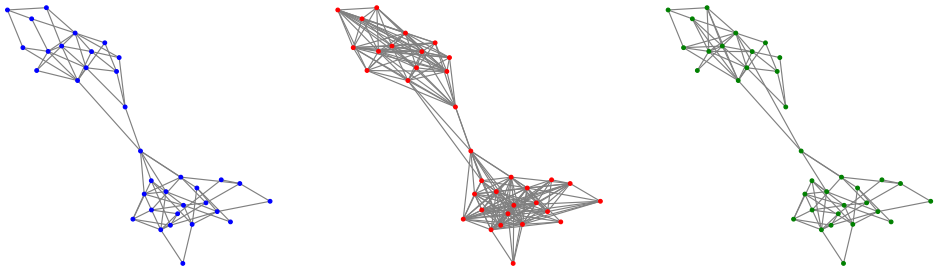


(c) Sample from the tree validation set

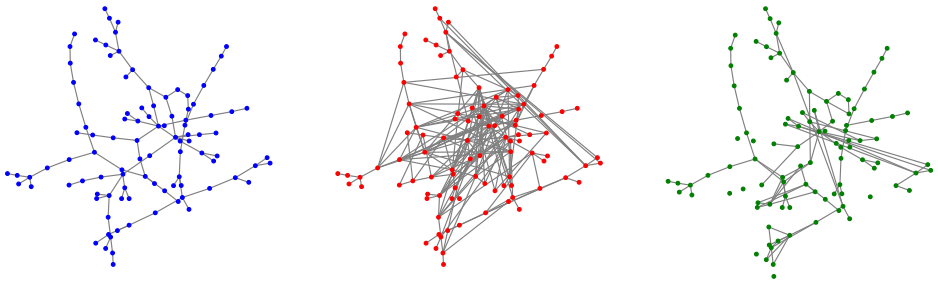
Figure A.4: Reconstructions of the validation graphs by the VQ-VAE. From left to right: ground truth (blue), reconstruction in VL (red), reconstruction in TR (green).



(a) Sample from the lobster training set



(b) Sample from the SBM training set



(c) Sample from the tree training set

Figure A.5: Reconstructions of the training graphs by VQ-VAE. From left to right: ground truth (blue), reconstruction in VL (red), reconstruction in TR (green).

A.2 Generation

A.2.1 Scores

Validation Set

AE	Disc.	Temp.	Degree	Clustering	Spectral	Ratio	Valid	Novelty
TR	DAB	CST	6.96E-02	1.11E-13	1.37E-01	2.17E+02	**0.29**	1.00
TR	DAB	DMM	8.01E-02	6.02E-14	1.41E-01	2.43E+02	0.28	1.00
TR	DAD	CST	3.37E-02	3.45E-12	3.72E-01	2.37E+02	0.08	1.00
TR	DAD	DMM	3.80E-02	3.52E-12	3.79E-01	2.50E+02	0.03	1.00
VL	DAB	CST	3.03E-02	6.47E-13	1.25E-01	**1.23E+02**	0.05	1.00
VL	DAB	DMM	2.07E-02	1.14E-12	2.01E-01	1.34E+02	0.02	1.00
VL	DAD	CST	2.71E-02	2.13E-12	2.32E-01	1.62E+02	0.03	1.00
VL	DAD	DMM	5.55E-02	4.69E-12	5.12E-01	3.47E+02	0.11	1.00

Table A.1: Results of graph generation on the validation set of the lobster dataset.

AE	Disc.	Temp.	Degree	Clustering	Spectral	Ratio	Valid	Novelty
TR	DAB	CST	5.17E-02	5.29E-02	2.92E-01	9.14E+01	0.00	1.00
TR	DAB	DMM	5.18E-02	6.58E-02	2.61E-01	**8.84E+01**	0.00	1.00
TR	DAD	CST	3.41E-01	2.70E-02	1.29E-01	3.96E+02	0.00	1.00
TR	DAD	DMM	1.16E-01	2.70E-02	6.19E-02	1.37E+02	0.01	1.00
VL	DAB	CST	6.58E-02	1.32E-01	2.27E-01	1.02E+02	0.00	1.00
VL	DAB	DMM	8.28E-02	2.99E-02	1.85E-01	1.14E+02	0.00	1.00
VL	DAD	CST	1.86E-01	2.70E-02	6.92E-02	2.16E+02	0.00	1.00
VL	DAD	DMM	9.49E-02	3.02E-02	1.65E-01	1.25E+02	0.00	1.00

Table A.2: Results of graph generation on the validation set of the SBM dataset.

AE	Disc.	Temp.	Degree	Clustering	Spectral	Ratio	Valid	Novelty
TR	DAB	CST	6.52E-02	5.49E-03	4.16E-01	5.07E+02	**0.35**	1.00
TR	DAB	DMM	4.15E-02	1.55E-03	4.10E-01	**2.71E+02**	0.14	1.00
TR	DAD	CST	2.30E-01	1.01E+00	1.96E-01	3.47E+04	0.00	1.00
TR	DAD	DMM	2.44E-01	5.03E-12	5.42E-01	1.11E+03	0.05	1.00
VL	DAB	CST	1.19E-01	2.40E-03	4.31E-01	6.40E+02	0.11	1.00
VL	DAB	DMM	1.40E-01	3.96E-03	3.78E-01	7.76E+02	0.17	1.00
VL	DAD	CST	3.52E-01	6.00E-12	6.01E-01	1.59E+03	0.11	1.00
VL	DAD	DMM	2.00E-01	3.55E-02	5.10E-01	2.10E+03	0.11	1.00

Table A.3: Results of graph generation on the validation set of the tree dataset.

Training Set

AE	Disc.	Temp.	Degree	Clustering	Spectral	Valid	Novelty
TR	DAB	CST	3.09E-01	1.39E+00	3.06E-01	0.00	1.00
TR	DAB	DMM	1.76E-01	5.80E-01	5.54E-01	0.06	1.00
TR	DAD	CST	8.31E-02	5.91E-12	6.62E-01	0.03	1.00
TR	DAD	DMM	5.35E-02	5.56E-02	5.97E-01	0.05	1.00
VL	DAB	CST	3.95E-02	1.93E-12	2.33E-01	0.23	1.00
VL	DAB	DMM	3.45E-02	1.50E-02	1.54E-01	0.14	1.00
VL	DAD	CST	4.01E-01	1.02E+00	5.88E-01	0.00	1.00
VL	DAD	DMM	1.06E-01	1.03E+00	1.04E-01	0.00	1.00

Table A.4: Results of graph generation on the training set of the lobster dataset.

AE	Disc.	Temp.	Degree	Clustering	Spectral	Valid	Novelty
TR	DAB	CST	1.28E-01	3.91E-02	1.33E-01	0.00	1.00
TR	DAB	DMM	1.08E-01	6.16E-02	2.56E-01	0.00	1.00
TR	DAD	CST	3.13E-01	1.88E-02	1.19E-01	0.00	1.00
TR	DAD	DMM	1.55E-01	1.88E-02	6.25E-02	0.00	1.00
VL	DAB	CST	1.37E-01	5.17E-01	2.26E-01	0.00	1.00
VL	DAB	DMM	1.57E-01	1.88E-02	7.40E-02	0.00	1.00
VL	DAD	CST	2.93E-01	1.89E-02	1.17E-01	0.00	1.00
VL	DAD	DMM	1.44E-01	3.93E-02	2.03E-01	0.00	1.00

Table A.5: Results of graph generation on the training set of the SBM dataset.

AE	Disc.	Temp.	Degree	Clustering	Spectral	Valid	Novelty
TR	DAB	CST	1.02E-01	8.15E-01	2.89E-01	0.03	1.00
TR	DAB	DMM	2.51E-01	3.95E-01	4.33E-01	0.05	1.00
TR	DAD	CST	6.34E-01	1.01E+00	5.00E-01	0.00	1.00
TR	DAD	DMM	7.11E-01	1.51E+00	6.56E-01	0.00	1.00
VL	DAB	CST	1.81E-01	1.18E+00	2.96E-01	0.01	1.00
VL	DAB	DMM	7.69E-02	1.03E+00	2.58E-01	0.00	1.00
VL	DAD	CST	7.82E-01	2.00E+00	6.59E-01	0.00	1.00
VL	DAD	DMM	7.26E-01	1.07E+00	6.10E-01	0.00	1.00

Table A.6: Results of graph generation on the training set of the tree dataset.

A.2.2 Visualization

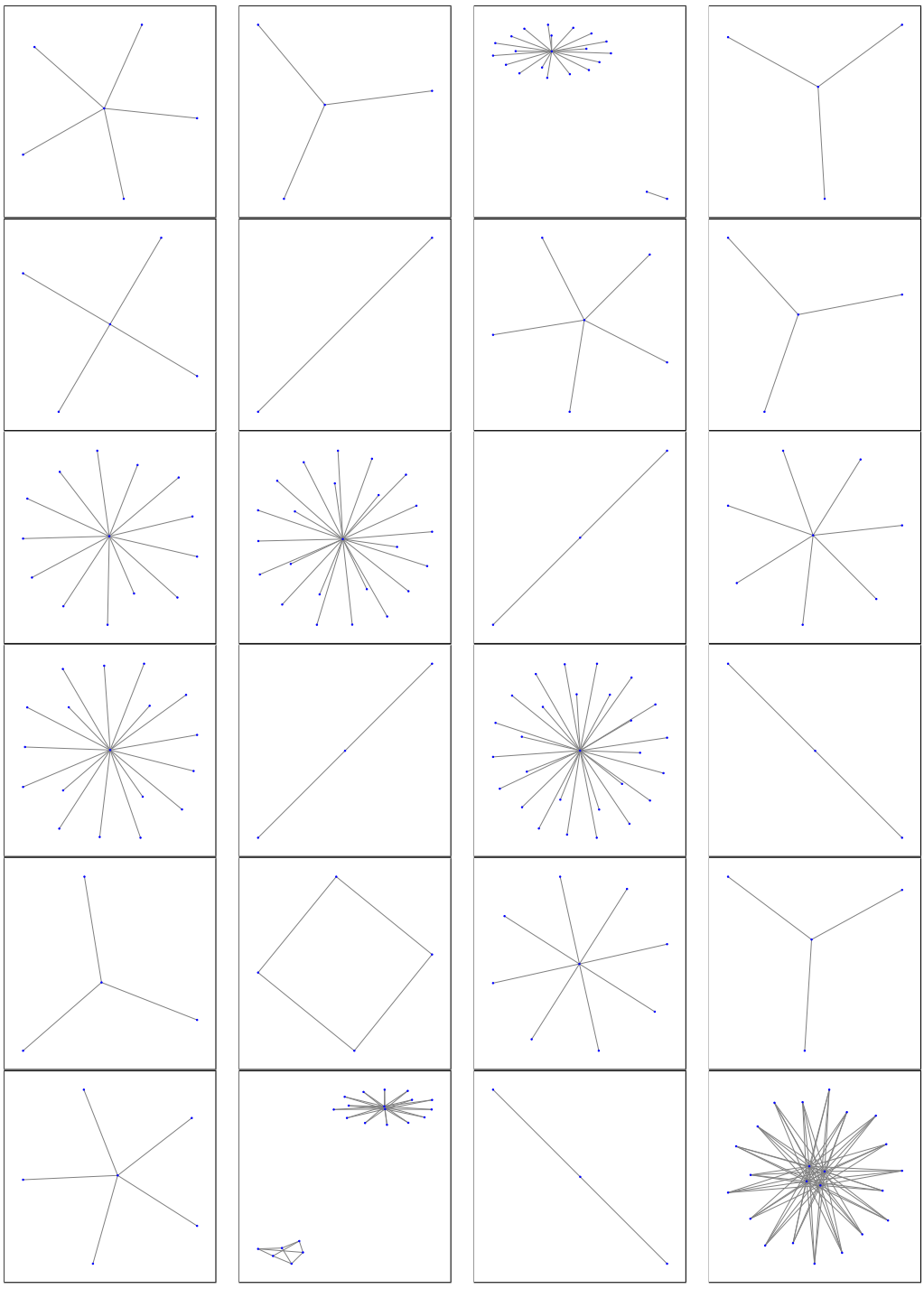


Figure A.6: Examples from the generated random lobster graphs.

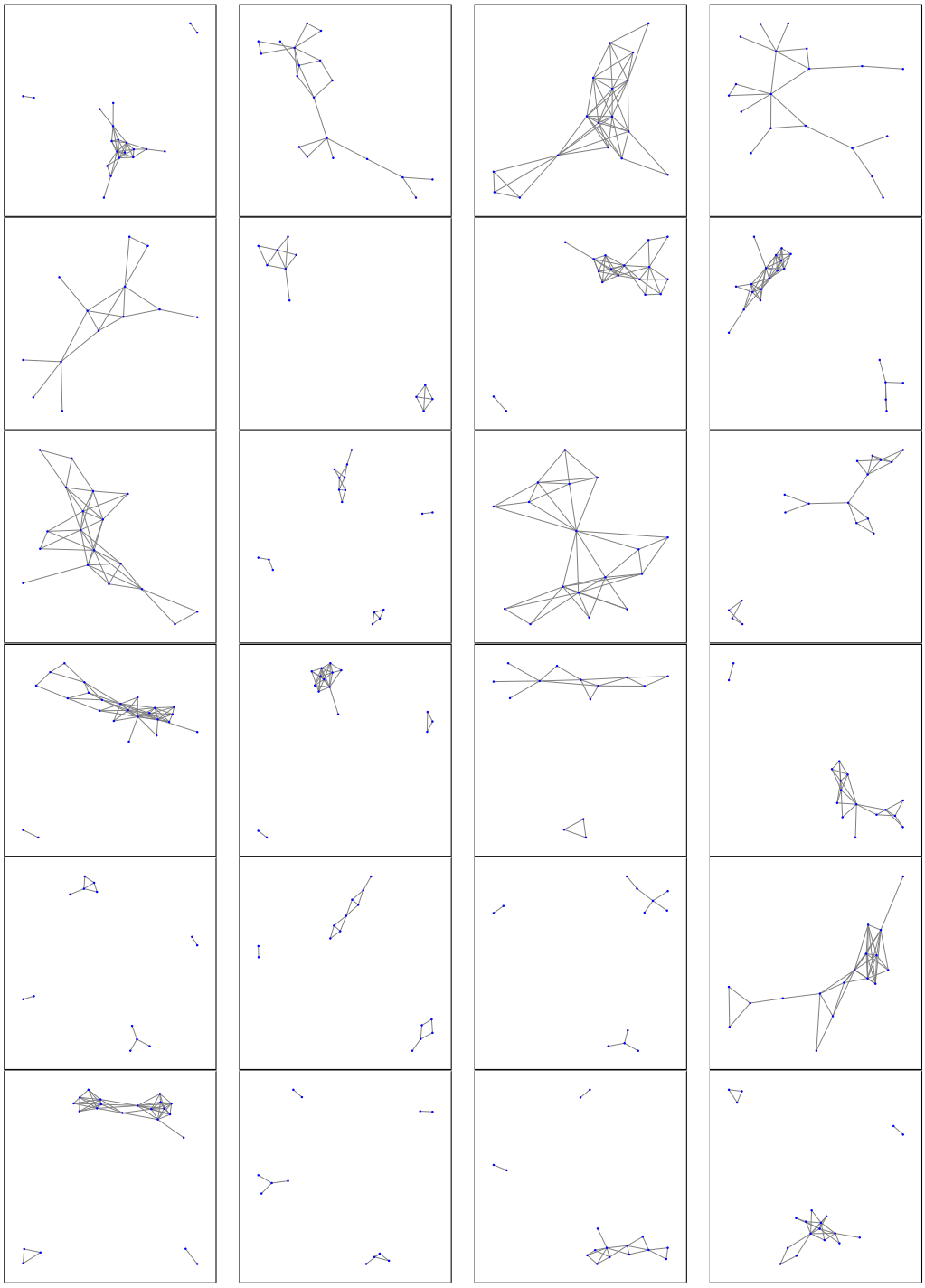


Figure A.7: Examples from the generated random SBM graphs.

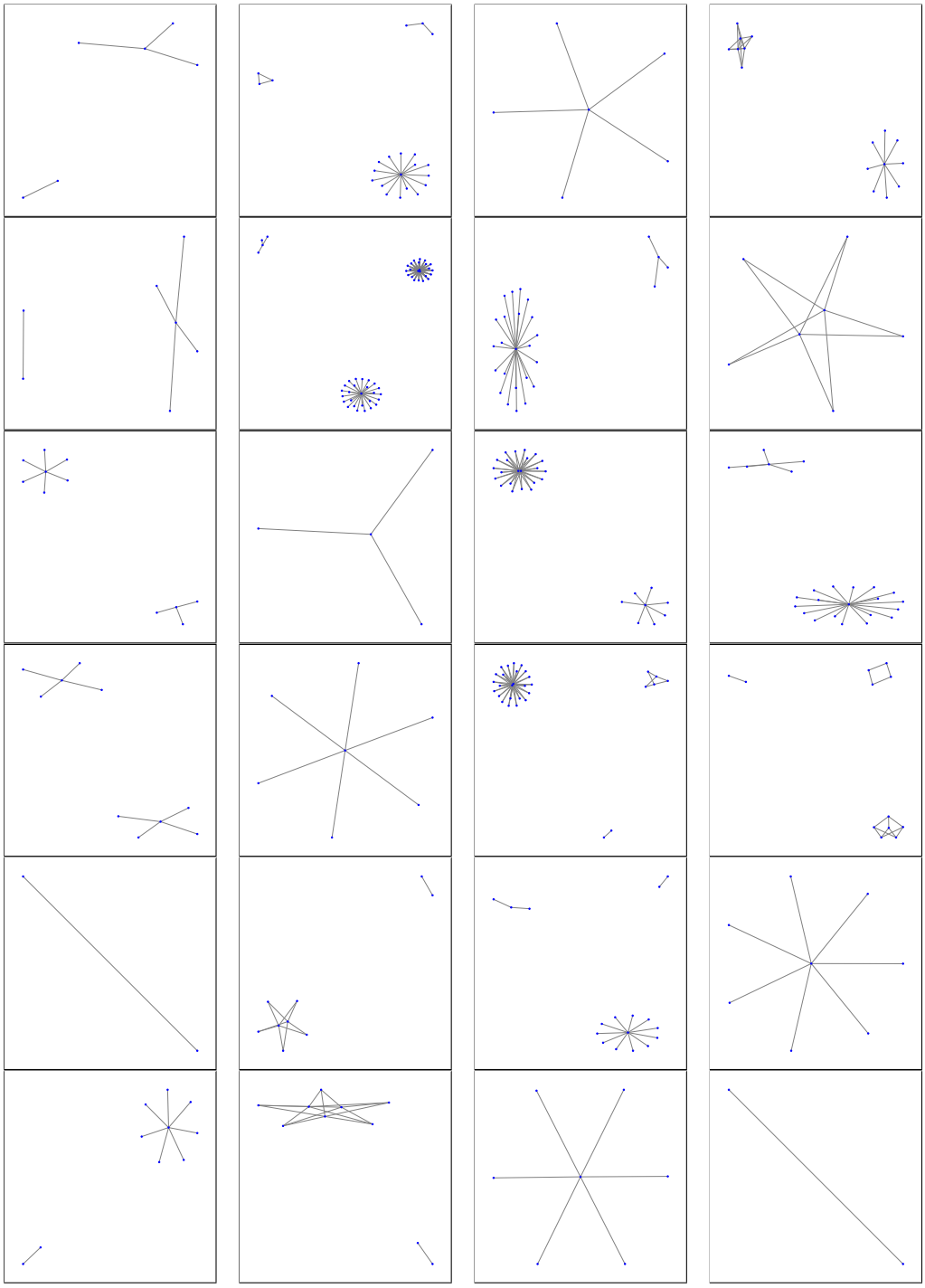


Figure A.8: Examples from the generated random tree graphs.

Appendix B

Hyperparameters

Table B.1: Hyperparameters for the VQ-VAE

Symbol	Name	Value	Description
d_z	embedding dimension	512	dimension of the node embeddings
$\|\mathcal{C}\|$	# codebook elements	256	number of elements in the dictionary
τ_0	initial temperature	2.0	initial temperature for the Gumbel-Softmax distribution
τ_∞	final temperature	0.2	final temperature for the Gumbel-Softmax distribution
α	cooling rate	0.9998	multiplier of temperature at each epoch
d_{E_θ}	encoding dimension	128	hidden dimension of GIN encoder
$\|\mathcal{L}(E_\theta)\|$	# encoder layers	20	number of layers in the GIN encoder
p_E	encoder dropout	0.7	probability of dropout for each encoder layer
$\mathbf{d}_{\Sigma_D}$	s2s dimensions	{512, 256, 128}	dimensions of layers in the set-to-set network
$\mathbf{d}_{\Gamma_D}$	g2g dimensions	{256, 64, 16, 4}	dimensions of layers in the set-to-set network
λ_{kl}	KLD scale	0.0005	scale of the KL divergence term for VQ-VAE

Table B.2: Hyperparameters for the GAN

Symbol	Name	Value	Description
d_z	embedding dimension	512	dimension of the node embeddings
$\|\mathcal{C}\|$	# codebook elements	256	number of elements in the dictionary
$\|\mathcal{L}(G_\gamma)\|$	# generator layers	2	number of layers in the generator
h_{G_γ}	# heads in generator	8	number of heads in the generator
$\|\mathcal{L}(F_{\text{TrEnc}})\|$	# transformer layers of discr.	2	number of layers in the GIN encoder
$\|\mathcal{L}(F_{\text{GIN}})\|$	# GIN layers of discr.	10	number of layers in the GIN of discriminator
$d_{F_{\text{GIN}}}$	GIN dim of discr.	128	dimension of GIN in discriminator
$\mathbf{d}_{F_{\text{MLP}}}$	MLP dims of discr.	{512, 128, 32, 8, 2}	dimensions of MLP in discriminator
h_{F_ψ}	# heads in discr.	8	number of heads in the discriminator
$p_{F_{\text{GIN}}}$	GIN dropout of discr.	0.7	probability of dropout for each GIN layer of discriminator
$p_{F_{\text{MLP}}}$	MLP dropout of discr.	0.25	probability of dropout for each MLP layer of discriminator
$\mathcal{P}$	multi-pooling operations	{`sum`, `max`, `min`, `std`}	operations to perform on the node embeddings
λ_{gp}	gradient penalty scale	5.0	scale of the gradient penalty term for WGAN

Appendix C

Additional Studies

C.1 Gaussian Distribution Generation

In order to evaluate the GAN independently from the VQ-VAE, we work on a simpler problem of generating a Gaussian distribution shape on vectors. This problem is selected regarding that the outputs we desire from the generator are also categorical distributions. We use the same GAN architecture with the discriminator at bottleneck configuration but we do not include the Wasserstein gradient penalty during the training. In other words, we consider each sample as a one-node graph. Moreover, we also try using an MLP instead of a transformer encoder for the generator.

We use a dataset consisting of Gaussian probability vectors with a support of $[-7.0, 7.0]$ with a size of 512, which is the number of codebook elements in the original experiments. The mean and standard deviation of the Gaussians are selected uniformly in $[-3.0, 3.0]$ and $[0.1, 3.0]$, respectively. As a success measure, we use the KL divergence between the generated vector and the corresponding Gaussian distribution of the same mean and variance.

At the end of 183 runs with a ranging number of epochs and dataset sizes, we find out that the transformer encoder is a better choice for the generator. A visualization of the generated Gaussians by the best run in terms of the KL divergence in Fig. C.1. For comparison, the results in early epochs and in the case of a mode collapse are given in Fig. C.2 and C.3, respectively.

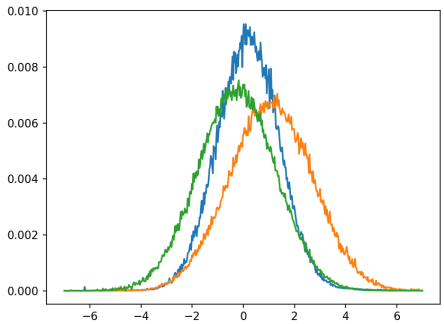


Figure C.1: Generated 3 Gaussian distributions by the best run in terms of the KL divergence.

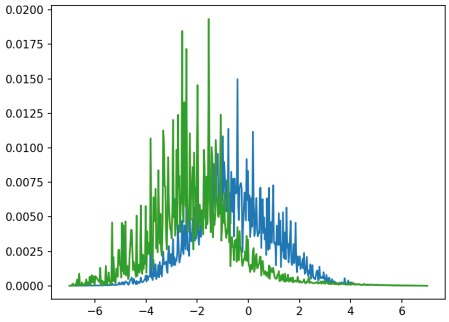


Figure C.2: Generated 3 Gaussian distributions in the early epochs.

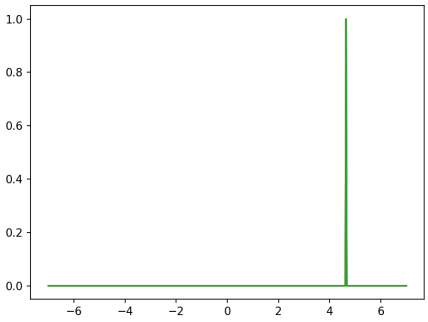


Figure C.3: Generated 3 Gaussian distributions in the case of mode collapse.

Appendix D

Technical Details

The scripts for the experiments were written in the Python language of version 3.9.7. The important packages used throughout the project and their versions can be found below:

- PyTorch [45] version: 1.10.1
- Cuda Toolkit [46] version: 11.6 (Local), 11.3 (Remote)
- PyTorch Cuda [45] version: 11.3
- PyTorch Geometric (PyG) [47] version: 2.0.2
- NumPy [48] version: 1.22.3
- NetworkX [43] version: 2.8

The experiments were conducted on Linux machines with Tesla P100-PCIE-16GB GPU provided by the Swiss National Supercomputing Centre (Italian: *Centro Svizzero di Calcolo Scientifico*, CSCS). The total computational power used for the experiments was 4212.3 node hours.

The results are stored and analyzed via Weights & Biases.