UNIVERSITY OF
CAMBRIDGE
Department of Computer
Science and Technology

Reimagining Graph Topology: Exploring Variational and Attentional Approaches

Teodora Rëu

Churchill College

June 2023

Submitted in partial fulfillment of the requirements for the
Master of Philosophy in Advanced Computer Science

Total page count: 60

Main chapters (excluding front-matter, references and appendix): 48 pages (pp 7–54)

Main chapters word count: 14206

Methodology used to generate that word count:

[

```
$ make wordcount
gs -q -dSAFER -sDEVICE=txtwrite -o - \
   -dFirstPage=7 -dLastPage=53 9651M.pdf | \
egrep '[A-Za-z]{3}' | wc -w


14206
```

]

Declaration

I, Teodora Rëu of Churchill College, being a candidate for the Master of Philosophy in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: Teodora Rëu

Date: 1 June 2023

Abstract

Graph Neural Networks (GNNs) have demonstrated impressive achievements in learning from data structured as graphs. However, when the learning task involves capturing long-range interactions, GNNs do not naturally address the phenomenon known as over-squashing, which can result in sub-optimal performance. Over-squashing is particularly noticeable in datasets with long-range and higher-order interaction between distant nodes. To address over-squashing, the literature has proposed graph rewiring as a potential solution. However, among the various methods proposed in the literature, none are data-driven and effectively consider both edge addition and removal. This project investigates the impact of over-squashing on graph-based tasks and explores differentiable rewiring methods. Two pipelines based on Variational Graph Autoencoder (VGAE) and Graph Transformer (GT) architectures are examined. Firstly, by incorporating the spectral gap together with a coefficient into the loss function of a VGAE we are able to tune the connectivity of the proposed adjacency matrix so that more long-range interaction edges are going to be built. Moreover, Graph Transformers work by explicitly considering all pairwise interactions between the graph's nodes, and learning *soft* adjacency matrices, which can be found in the attention mechanism. We extract the attention coefficient matrices and analyze them from a spatial and spectral perspective to understand their distribution and characteristics. Our findings can be summarized in the following way. By using VGAE we were able to build new rewirings for our graph which obtained strong empirical results comparable to Graph Transformers. For transformers, we discovered that the attention coefficient matrices and master nodes (nodes that are connected to every other node) seem to be the key mechanism to employ long range-interactions.

Acknowledgements

First and foremost, I would like to express my gratitude to my two supervisors, Challenger and Pietro, and my co-supervisor Chaitanya, for their invaluable guidance and insightful feedback throughout this project. I am also deeply grateful to Lorenzo and Francesco for our numerous research meetings, the exchange of ideas, and their patience in explaining the intricacies of the state-of-the-art.

This year has been truly amazing for me. Prior to joining this program, I had only a handful of friends whom I could rely on. Now, the circle of friends has expanded, and it even includes individuals with overlapping names. Not only have they provided me with emotional support, but they have also have been make my days brighter with fascinating conversations and moments I will never forget. I would like to extend my heartfelt thanks to Sergiu, Peter, Sarah, Theo, Harry, Jordan, Reece, Lena, Jeremy, Rowan, George, Pietro, Maria-Sofia, Lorenzo, Alexandra, Francesco D, Francesco C, and Francesco P. Together, you have created an environment where I could thrive and reach new heights.

I am immensely grateful to my parents for their unwavering support throughout my life, especially during my formative years as a very different kind of kid. Additionally, I want to express my appreciation to my partner, George, for standing by me during the challenging moments when this MPhil pushed me to my limits.

Lastly, I dedicate this work to the exceptional mathematician Vicky Neale[†], who, five years ago, gave me with the opportunity of a lifetime to fall irreversibly in love with research.

Contents

Chapter 1

Introduction

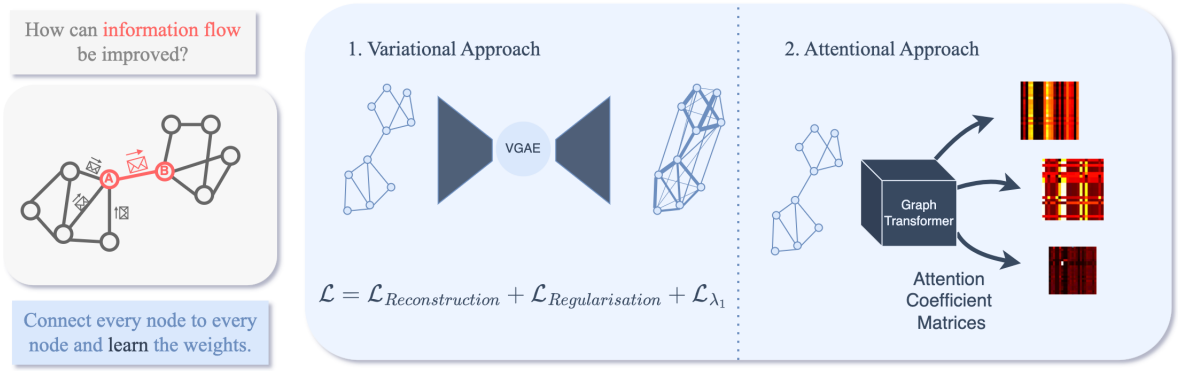

Figure 1.1: This project takes two approaches to understand over-squashing and rewiring. By rewiring we mean altering graph structure by building new weighted edges. Firstly, we study how incorporating the spectral gap (λ_1) into the loss function of a Variational Graph Autoencoder improves graph connectivity, leading to enhanced information flow and alleviation of over-squashing. Secondly, we investigate the attentional mechanisms that are responsible for the effective modelling of long-range interactions in state-of-the-art transformers.

Graph Neural Networks (GNNs) have proven to be a powerful tool for learning graph-structured data, as evidenced by their success in various domains such as social networks [55, 28], traffic forecasting [17], and molecular biology [46]. Typically GNNs operate in the message-passing paradigm by exchanging information between nodes that are connected by an edge, giving rise this way to a class of Message-Passing Neural Networks (MPNNs) [25]. While this class of models has been shown to have successes, MPNNs have their limitations varying from low expressive [53, 41] to over-smoothing [42, 12, 8].

One of the main challenges that the field faces currently, arising mostly in inductive graph tasks (ex. graph classification or regression), is over-squashing [2]. That is defined as an issue occurring when MPNNs propagate messages across distant nodes, with the exponential expansion of messages over neighbourhoods getting *squashed* into a fixed-size vector. This problem becomes even more challenging on graphs with high diameters, such as those met in computational chemistry problems (ex. Peptides [22]). Emphasizing and

considering long-range interactions can significantly enhance performance on tasks that rely on such interactions.

To address this issue, the literature proposes a solution called graph rewiring [49], which involves altering the topology of the graph by adding or removing edges. Most literature at this point focuses on non-differentiable, algorithmic rewiring, such as connectivity diffusion [24], evolution [47], adding new bridges [6], and multi-hop filters [7, 23]. However, all these solutions are algorithmic-based solutions, they are not data and task-driven. A differentiable solution has been proposed by [3]. However, their solution only removes edges from the initial adjacency matrix, not considering adding edges.

Recent advancements in the field of rewiring focus on defining over-squashing in terms of bottlenecks within a graph [2], by using spectral metrics. They utilise concepts such as spectral gap λ_1, commute times, and Cheeger constant to quantify over-squashing [3, 18]. The spectral gap λ_1, which corresponds to the first strictly positive eigenvalue of the graph Laplacian, is a measure of community structure within a graph. It also serves as an upper bound for graph metrics that are challenging to compute but can measure over-squashing or bottlenecks, such as commute times, required energy [39], and Cheeger constant [47].

We believe that the field misses a differentiable spectral direction, in which edges are both added and removed, based on spectral metrics. Moreover, a direction that focuses on rewiring in which the proposed adjacency matrices are fully connected graphs with weighted edges, where the weight of an edge represents in what proportion the information from one node will get passed to its neighbour.

Our Contributions: This work focuses on understanding over-squashing and its impact on graph-based tasks. We are interested in differentiable rewiring methods, and we explore two pipelines: an architecture which is Variational Graph Autoencoder (VGAE) [35] based, and another which is Graph Transformer [19] based. The evaluation is conducted on the ZINC [20] and Peptides-struct datasets [22], providing insights into the significance of graph topology in graph learning pipelines. The ZINC dataset emphasizes learning local structure, while the Peptides dataset requires capturing long-range interactions between distant nodes. The main contributions of this work can be summarized as follow:

A Variational Approach: The spectral gap λ_1 is the first positive eigenvalue of the graph Laplacian and also a measure for connectivity in a graph. It is both feasible to compute and it also plays a central role in recent research on rewiring and measuring bottlenecks, which is believed to be the main cause of over-squashing [47, 18, 3]. We use a Variational Graph Autoencoder (VGAE) architecture to generate rewired adjacency matrices that alleviate over-squashing by including the spectral gap $\mathcal{L}_{\lambda_1}$ of the generated graph in the loss function. We argue that the obtained adjacency matrices will contain

useful information and better propagation of the information will help on supervised tasks. We address the following questions: (1) can VGAE-based models learn to build such graphs in which bottlenecks are ameliorated? (2) is the obtained rewiring helpful to a supervised learning pipeline in endorsing long-range interactions? (3) does alleviating bottlenecks actually help in long-range interaction supervised tasks?

We utilized the Variational Method to investigate the generation of new fully connected weight matrices using VGAE. In this approach, we introduced a new term to the loss function, denoted as $\mathcal{L}^{\alpha}_{\lambda_1} = \alpha\lambda_1(A^*)$. Here, A^* represents the rewired adjacency matrix, and α is a coefficient. By incorporating $\mathcal{L}^{\alpha}_{\lambda_1}$ to the loss, we were able to control the connectivity of the generated graphs. This allowed us to control the sparsity of the generated graphs. We conducted multiple experiments on both the Peptides-struct and ZINC datasets, employing various combinations of layers and positional encodings. We evaluated the performance of these experiments using different sets of generated adjacency matrices, and our findings are:

(1) That VGAEs are able to learn such graphs in which the bottlenecks are ameliorated.

(2) Our empirical findings demonstrated the value of our generated rewirings, particularly in the case of long-range interaction datasets such as Peptides, where our models performed almost as well as Transformer based architectures.

(3) We generated graph rewirings which were expressing big bottlenecks and graph rewirings which were did not notice major differences in the performance over test sets in Peptides and ZINC.

An Attentional Approach: Graph Transformers are a type of architecture that considers all to all message passing and learns *soft* adjacency matrices through the attention mechanism [32, 19]. On long-range interaction-based tasks [22], and also graph benchmarking [20], transformers obtain state-of-the-art results. The attention coefficient matrices are learned by the model and vary from one graph to another. This makes the attentional mechanism a differentiable method for rewiring. In order to gain a deeper understanding of the success of GTs, we analyse the attention coefficients matrices that transformers learn while performing a supervised task, by computing topological graph metrics over the matrices. We argue that in the attention heads, at each layer, transformers actually learn new rewirings for the graphs. We want to answer the following questions: (1) how do these attention coefficient *adjacency matrices* look, (2) are they correlated at all with the initial adjacency matrix of the graph, (3) do they encode long-range interactions? We strongly believe that exploring these architectures can reveal how important is the topology of the graph in graph learning pipelines. In order to answer these questions we are going to compute spectral gap, diameter, and commute times over the attention coefficient metrics to uncover the recipe that leads to state-of-the-art results in the case of transformers.

We extract the attention of two different models, one that uses convolution on the initial adjacency matrix, and one that only does message passing on the fully connected adjacency matrix. Our findings can be summarized as follows:

(1) The main mechanism behind transformer attention is *master nodes* (nodes that are connected to every other node).

(2) The answer to the second question varies depending on the task. In the case of ZINC, we observe some correlation between the attention and the initial adjacency matrix, which aligns with the task's structure-based nature. However, for Peptides, we find that the attention coefficient matrices do not exhibit a significant correlation with the initial adjacency matrix. This suggests that they do not primarily capture the graph structure, but rather express stable forms of long-range interactions.

(3) By analyzing the coefficient adjacency matrices using metrics for long-range interactions such as diameters, spectral gap, and commute times, we discover that for Peptides, the attention extracted from the first and last layers forms nearly fully connected graphs, where the weights on edges are equal. In contrast, the attention coefficient layers for ZINC exhibit a more balanced structure.

Chapter 2

Related work

In this section, we provide an overview of the relevant literature that addresses the challenges of over-squashing, and under-reaching in MPNNs through graph rewiring.

Graph Rewiring *Rewiring* is the process of changing the graph adjacency matrix to control the information flow, and improve the connectivity of a graph. Recent approaches to address over-squashing in MPNNs share a common idea: replacing the original graph G with a rewired graph R(G) that exhibits improved connectivity [49]. When rewiring, one needs to pick a metric to understand if the rewired graph is *better* or not. And by better we mean that the metric of choice shows improvement, in the sense that if f is our metric then we would have $f(G) > f(R(G))$ (or $f(G) < f(R(G))$ depending on the nature of the metric of choice). There are many types of metrics and possible choices for the approach, we will take the classification in [18], which said methods can be either spatial (improving metrics such as diameters) or spectral (improving metrics such as λ_1).

Spatial methods Involve replacing G with R(G) in such a way that the diameter of R(G) is much smaller than that of G. This can be achieved by explicitly adding edges (potentially attributed) between distant nodes or enabling communication between distant nodes. The work of [11] uses additional nodes and edges which are augmented with positional encodings, making this way their method model-agnostic. In [9] the authors introduce a new type of rewiring between higher-order structures like cellular complexes. Other methods use k-hop matrices, which minimises the diameter in an indirect way [1, 27]. We introduce in this category the work of [47] as well since they use Ricci curvature to understand over-squashing.

Spectral Methods Spectral methods involve *rewiring* techniques that aim to maximize or minimize metrics related to spectral values, such as the spectral gap (λ_1), the Cheeger constant, and Commute Times. Several works in this domain utilize different approaches to improve information flow. In the work of [3], they employ the Lovász bound, which refers to Commute Times, to improve information flow. Other

works, such as [16], use graph expanders to enhance information flow by building a graph which is independent of the input. [33], [4] utilize the spectral gap in their pipeline to improve information flow.

Another classification of these methods can be based on whether they are algorithm-based or data-driven based:

Algorithm based Most methods mentioned until now [11, 1, 47, 16, 33, 4, 9] employ algorithms or some kind of processing of the data to create new adjacency matrices.

Differentiable The single differentiable method mentioned is DiffWire employed by [3], which is limited because it only removes edges from the initial graph when rewiring.

Graph Transformers Graph Transformers represent an extreme example of rewiring [36, 40, 43, 54], where R(G) is a complete graph with learned weighted edges computed by attention mechanism over all pairs of nodes. We are not aware of any study of the attention coefficient matrices from a long-range interaction point of view.

2.1 Motivation

In this section, we justify graph rewiring as a potential solution for current problems in the field. As previously mentioned in Section 2 there are no differentiable methods that both remove and add edges. The work of [3] is differentiable but is concerned only with removing edges. To employ differentiable and data-driven graph rewirings we take our two approaches which both add and remove edges. Next, we will motivate our choice of architecture.

Why Variational Autoencoder for Graph Rewiring? There are many types of generative models: Variational Autoencoder [35], Generative Adversarial Neural Networks [15] and Denoising Diffusion Probabilistic [29]. We think that Variational Graph Autoencoder could represent a good framework for rewiring because it is a framework that allows the distribution of the generated objects to be different from the distribution of the data, and because they have been before successfully in inductive tasks [52, 38, 26, 56]. Besides reconstruction error (i.e., which quantifies how similar the generated graph is with the initial graph) and regularisation error (i.e., which quantifies how normally distributed the hidden dimension is), we can promote the generation of graphs that mitigate the over-squashing phenomena by adding a penalty term to the loss. In this way, we can control the output graph to not be very different from the initial adjacency matrix, by using coefficients for each loss element. Another reason to do so is that Variational Autoencoders are an unsupervised architecture which means that the graph rewiring they

proposed will be based only on the initial topology of the graph and the node features, without the graph labels or regression values.

Another option for a generative model could have been Denoising Diffusion Models, so it is crucial to motivate the selection of a variational autoencoder-based model over a denoising diffusion-based architecture, which is currently quite popular. While the Denoising Diffusion Probabilistic Model (DDPM) [29] has recently gained significant attention for its ability to generate data, graph adjacency matrices included [31], it has a fundamental limitation: the generated objects must have the same distribution as the data from which they were learned. Therefore, if we learn from a dataset of adjacency matrices, we cannot expect the generated graphs to have a better rewiring. This issue is evident in score matching-based models [45, 48], where the loss function approximates the score $\nabla \log p(x(t))$ using $\nabla \log p(x(0)|x(t))$, indicating that the distribution of our data at time t must be the same as that of our data at time 0 times the noise that will take us to p(x(t)). Therefore, although very popular at this time, diffusion models are not appropriate for our idea. Similarly, Generative Adversarial Networks (GANs) [15] face a similar constraint. In GANs, the generative model is trained to produce samples that can deceive a discriminator, which acts as a trained classifier, into believing that they originate from the initial distribution.

Why explore the attention of Transformers? Due to remarkable results on various benchmarks, including Peptides and ZINC, [20, 22], we decided to explore the attention of Graph Transformers. Transformers achieve these results by establishing connections between every node and utilizing an attention mechanism that learns coefficients for node-to-node interactions at each layer. This mechanism helps alleviate bottlenecks and enables the encoding of long-range interactions. For example, in the case of Peptides, learning long-range interactions, and in the case of ZINC, learning structural encoding. We aim to study the landscapes of attention coefficients because we believe they contain valuable information. While Transformers have been widely used, the attention coefficient matrices have not been extensively investigated from a perspective that emphasizes long-range interactions. We believe that the information encoded in these matrices may provide insights for future directions in rewiring methods. To the best of our knowledge, long-range interaction hasn't been studied before on the attention coefficient for graphs.

Chapter 3

Background

This section provides the necessary background for the current work. It begins by introducing the concept of Message Passing Neural Networks (MPNNs) 3.1 and then introduces the problem which we are trying to solve which is **over-squashing** in Section 3.2. In 3.3, we present two types of positional encodings. The next sections then delve into the theoretical foundations of Graph Transformers 3.4 and Variational Graph Autoencoders 3.5. Furthermore, it presents relevant findings that highlight the significance of the spectral gap (λ_1) from a topological standpoint in Section 3.6.

3.1 Message Passing Neural Networks

We will adopt the definition from [10]. Let $\mathcal{G}(V, E)$ be a graph. The connectivity is encoded in the adjacency matrix $A \in \mathbb{R}^{n\times n}$, with $n = |V|$. Further, a node feature matrix $\mathbf{X} \in \mathbb{R}^{n\times k}$, giving features of the node u as $\mathbf{x}_u$. A message-passing GNN over this graph could be executed in the following way:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v\in\mathcal{N}_u} \psi\left(\mathbf{x}_u, \mathbf{x}_v\right)\right) \tag{3.1}$$

where $\psi : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}^m$ is a *message function*, $\phi : \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}^l$ is a *readout function*, and $\bigoplus$ is a permutation-invariant aggregation (sum or max). Both ψ and ϕ can be realised as MLPs.

This form of message-passing has several limitations. For instance, it fails to recognize simple substructures, as MPNNs are unable to distinguish between a single 6-cycle and two 3-cycles [49]. Other limitations include over-smoothing [37] and over-squashing [2]. We have included a dedicated section in the Appendix, specifically Appendix A.2, which provides a comprehensive overview of the message-passing networks (MPNNs) that we will utilize in our study. Please refer to this section for detailed information on the MPNN

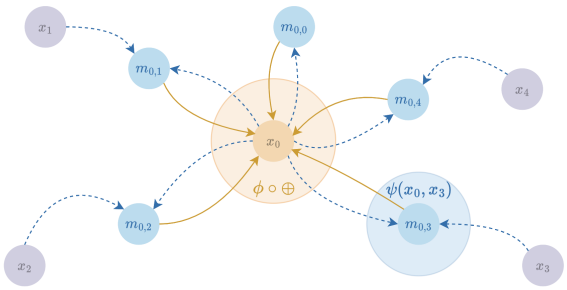

Figure 3.1: Pictorial overview of the message-passing scheme from Equation 3.1

types we have selected.

3.2 Introducing Over-Squashing

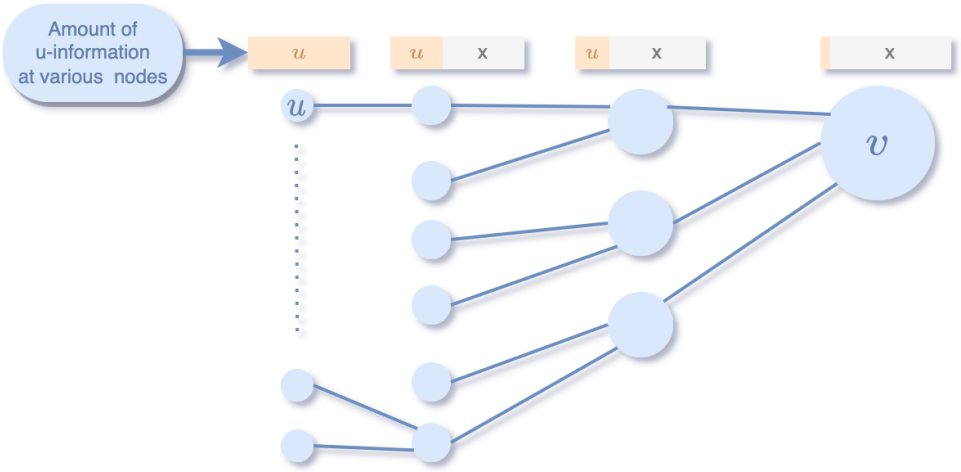

Figure 3.2: Over-squashing - We can see that the influence u has within other nodes decreases exponentially as it travels down to v. In very bottlenecked graphs over great distances, it is possible it vanishes completely, this is the intuition behind the work of [47]. With x we marked information or influence coming from other nodes.

In our work, we aim to introduce over-squashing in a manner similar to the approach presented by [18]. Over-squashing refers to the potential loss of information caused by the exponential increase in the number of messages transmitted over distance r in Message Passing Neural Networks (MPNNs).

In MPNNs, the aggregation of information from neighbouring nodes allows a node v to be influenced by features at a distance of r, requiring a minimum of r layers in the MPNN architecture [5]. However, the expansion of the receptive field in MPNNs can lead to over-squashing, where the excessive number of messages sent over distance r results in a potential loss of information [2].

The authors of [47] provided insights into the quantification of influence between nodes in MPNNs. They demonstrated that the decay of influence, represented by $(A^r)_{vu}$, where A

is the message-passing matrix and r is the distance between nodes v and u, is exponential with r. If $(A^r)_{vu}$ decays rapidly, it indicates that the feature of node v becomes less sensitive to the information contained at node u.

Additionally, [47] identified a relationship between over-squashing and edges with high negative curvature. However, it is important to note that their characterization specifically applies to the propagation of information up to 2 hops, and further investigation is required to understand the effects of over-squashing on information propagation beyond this distance.

3.3 Positional Encoding

In order to address the limited expressivity of Message Passing Neural Networks (MPNNs), various approaches have been proposed to augment the feature vector with additional information. These approaches include the utilization of positional encodings based on Random Walk diffusion and on Laplacian eigenvectors [21]. These methodologies aim to enhance the representation of nodes in a graph by incorporating neighbourhood information and have been previously used with many transformer-like architectures [36, 43, 21].

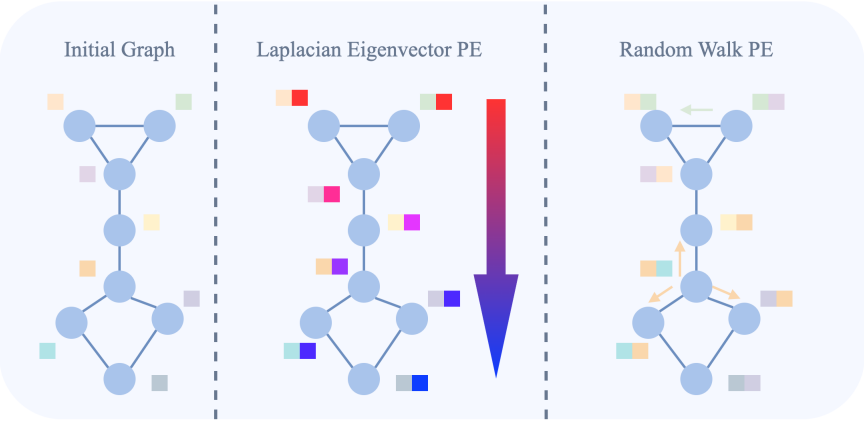

Figure 3.3: Exemplification of positional encoding methods with dimension 1. **Left:** The Initial Graph with initial features. **Middle:** In addition to their initial information, features are augmented with a value (in our case a colour) that resembles which community the node is coming from. For example, the nodes coming from the triangular shape situated at the top of the graphs the nodes will get the colour red, and for the community that comes from the lower part of the graph nodes will get the colour blue. **Right:** The features of the nodes are augmented with random information from one of their direct neighbours.

The Laplacian Eigenvector Positional Encoding transformation leverages the Laplacian matrix of the graph to compute the eigenvectors corresponding to non-trivial eigenvalues [21]. These eigenvectors are regarded as the Laplacian eigenvector positional encoding. By appending this encoding to the feature matrix, nodes gain a sense of their neighbourhood based on the eigenvector representation. This additional information enriches the

feature space and allows the model to capture more intricate patterns and relationships. As it can be seen in Figure 3.3, the Laplacian Eigenvector PEs added encode a distance between nodes.

The Random Walk Positional Encoding transformation incorporates positional encodings derived from Random Walk diffusion [21]. It follows a process where, starting from a given node, the graph is traversed through a random walk for a specified number of steps. At each step, the features of neighbouring nodes are aggregated, either by summing or taking the mean. The resulting aggregated information is then appended to the positional encoding vector. By employing this approach, the model captures collective knowledge and interactions from neighbouring nodes within a defined range. This inclusion of contextual information enhances the model's ability to consider the influence and relationships of neighbouring nodes at varying distances, with nodes with red features being far away from nodes with blue features. As it can be seen in Figure 3.3 this time nodes have got direct features coming from random 1-hop neighbours.

3.4 Graph Transformers

Graph Transformer Networks (GTNs) [43] apply the principles of Transformer architectures, originally designed for sequence data, to graph-structured data. The core idea of Transformers - capturing long-range dependencies through attention mechanisms - is extended to handle the irregular structure and variable size of graph data.

In the context of graphs, the attention mechanism is used to weigh the influence of each node's neighbourhood when updating its feature vector. This is akin to the self-attention mechanism in the original Transformer model but adapted to graph data.

Here is a simplified version of the update rule for a Graph Transformer layer, borrowing the notation from the original Transformer paper:

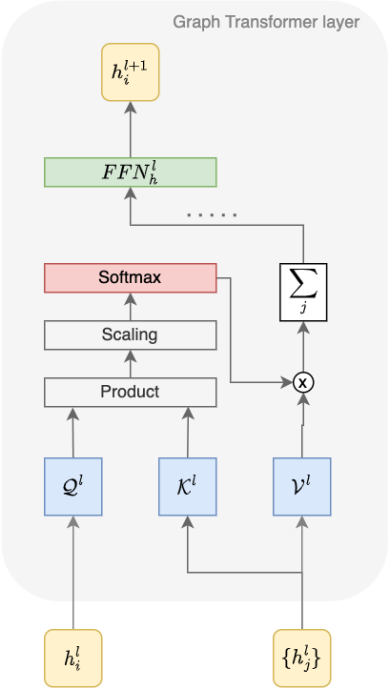

Figure 3.4: One layer of Graph Transformer architecture.

$$\mathbf{Z}_v = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \alpha_{vu} \mathbf{X}_u W \qquad (3.2)$$

$$\alpha_{vu} = \frac{\exp\left(\mathbf{a}^T \left[\mathbf{X}_v W_Q \parallel \mathbf{X}_u W_K\right]\right)}{\sum_{u' \in \mathcal{N}(v) \cup \{v\}} \exp\left(\mathbf{a}^T \left[\mathbf{X}_v W_Q \parallel \mathbf{X}_{u'} W_K\right]\right)} \tag{3.3}$$

Where $\mathbf{X}_v$ is the feature vector of node v; $\mathbf{Z}_v$ is the output feature vector of node v; W, W_Q, and W_K are learnable weight matrices, analogous to those in the original Transformer's self-attention mechanism; $\mathcal{N}(v)$ is the set of neighbours of node v; α_{vu} is the attention score that determines the influence of node u on node v; $\mathbf{a}$ is a learnable parameter vector. Here, the symbol $\parallel$ denotes concatenation.

In the first equation, each node v computes its new feature vector $\mathbf{Z}_v$ as a weighted sum of its neighbours' transformed feature vectors, with the weights given by the attention scores. In the second equation, the attention scores are computed using a softmax function, which allows the model to learn which nodes to pay attention to when updating each node's feature vector.

3.5 Variational Graph Autoencoder

Variational Graph Autoencoder (VGAE) [35] is an autoencoder architecture based. That is given some input data with N graph $\mathcal{G}(X, A)$ with $X \in \mathbb{R}^{N \times F}$ and $A \in \mathbb{R}^{N \times N}$ the autoencoder reduces the input to a lower hidden representation $h \in \mathbb{R}^{N \times f}$ by using GNN based layers, getting the lower hidden representation z_n, after which the. The Variational aspect of it comes from the fact that the hidden representations are forced to have a predefined distribution, which usually is the Gaussian distribution.

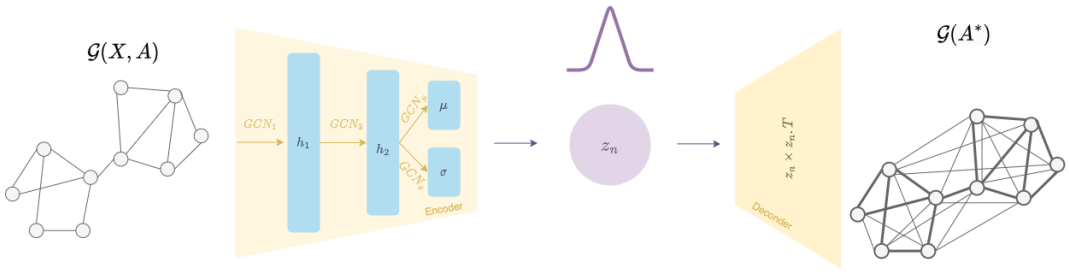

Figure 3.5: Variational autoencoder pipeline figure. The Encoder reduces our graph to a lower hidden representation z_n, which is forced by the regularisation loss to be normally distributed. Then a Decoder tries rebuilding the initial adjacency matrix. The reconstruction loss will penalize any major differences between the initial adjacency and the produced one.

In the work of Kipf and Welling [35], the encoder model is parameterized by an n-layered Graph Neural Network (GNN). It is important to note that any type of GNN layer, such as Graph Convolutional Networks (GCN) or Graph Isomorphism Networks (GIN), can be used in the encoding part:

$$q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q\left(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}\right), \text{ with } \quad q\left(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}\right) = \mathcal{N}\left(\mathbf{z}_i \mid \boldsymbol{\mu}_i, \operatorname{diag}\left(\boldsymbol{\sigma}_i^2\right)\right).$$

Here, $\boldsymbol{\mu} = \mathrm{GNN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A})$ represents the matrix of mean vectors $\boldsymbol{\mu}_i$, and similarly, $\log \boldsymbol{\sigma} = \mathrm{GNN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$. This corresponds to the Encoder in Figure 3.5. In their generative model, [35] define it using an inner product between latent variables:

$$p(\mathbf{A} \mid \mathbf{Z}) = \prod_{i=1}^{N} \prod_{j=1}^{N} p\left(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j\right), \text{ with } p\left(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j\right) = \sigma\left(\mathbf{z}_i^{\top} \mathbf{z}_j\right),$$

where A_{ij} are the elements of $\mathbf{A}$ and $\sigma(\cdot)$ is the logistic sigmoid function. It is not necessary that the decoder is deterministic, and in order to make it more expressive MLP layers can be used. The optimization of the variational lower bound $\mathcal{L}$ is performed with respect to the variational parameters present in the GNN layers:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A} \mid \mathbf{Z})] - \mathrm{KL}[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \| p(\mathbf{Z})]$$

In this loss function first part $\mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A} \mid \mathbf{Z})]$ is the reconstruction error, and what it does is it calculates the Cross-Entropy between A and A^* in Figure 3.5. The second argument $\mathrm{KL}[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \| p(\mathbf{Z})]$ calculates the KL divergence between the probability distribution of z_n and usually a Gaussian distribution $\mathcal{N}(0, \sigma^2 I)$.

3.6 The First positive eigenvalue of the Laplacian

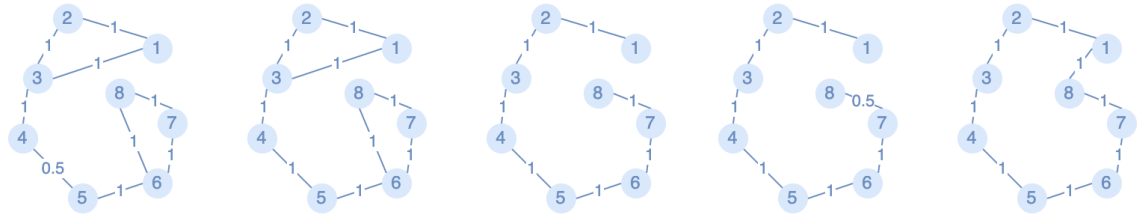

Figure 3.6: Intuition for λ_1 values as the weights of the edges change. In the first graph, the communities are clearly defined and easy to split so the λ_1 value is small. In the last picture, the community separation is not very clear anymore, and the energy to split the graph in two is higher so the λ_1 value increases. A not-so-intuitive value of λ_1 can be observed in the fourth graph, where λ_1 has increased because the two communities given by the areas around the edges of the graph have been weakened.

In this section, we present a formal introduction to the concept of rewiring in graphs and highlight the significance of λ_1 (the first strictly positive eigenvalue of the symmetric Laplacian) in this process. λ_1, also referred to as the spectral gap, plays a crucial role as it not only measures the level of clustering within a graph but also serves as an upper

bound for various important metrics associated with the concept of bottlenecks in a graph. We delve into several graph metrics that are closely linked to or effectively capture the bottleneck characteristics of a graph. Furthermore, we establish a compelling connection between these metrics and the value of λ_1, demonstrating their inherent interdependence.

Definition 1. *Consider a MPNN, a graph G with adjacency matrix A, and a map $\mathcal{R} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$. We say that $\mathcal{G}$ has been rewired by $\mathcal{R}$ if the messages are exchanged on $\mathcal{R}(G)$ rather than on G, with $\mathcal{R}(A)$ being the adjacency matrix of $\mathcal{R}(G)$.*

To address the issue of over-squashing, rewiring serves as a potential solution. When performing rewiring, there are two main approaches to consider: spatial methods and spectral methods. **Spatial methods** involve proposing $\mathcal{R}(G)$, aiming for a graph with a smaller diameter (i.e., the shortest path between the furthest nodes). This can be achieved by adding edges to connect the most distanced nodes in the graph [11]. By doing so, the graph becomes more interconnected, reducing the potential for over-squashing. On the other hand, **spectral methods** focus on minimizing metrics such as Effective Resistance, Commute Time, and Access Time. These metrics quantify different aspects of graph connectivity and flow dynamics. By minimizing these metrics through rewiring, the graph's bottleneck characteristics can be improved, enhancing its overall structure and avoiding over-squashing. We will introduce and explore these metrics in more detail, shedding light on their significance in the context of rewiring and mitigating over-squashing. In [39] the *Commute Time* or *Access Time* are introduced in the following way:

Definition 2. *The access time or hitting time H_{ij} is the expected number of steps before node j visited, starting from node i, with the following recursive formula between $i \neq j$ is:*

$$H(i,j) = 1 + \frac{1}{d(i)} \sum_{v\in\Gamma} H(v,j)$$

The sum

$$\kappa(i,j) = H(i,j) + H(j,i)$$

is called the commute time: this is the expected number of steps in a random walk starting at i, before node j is visited and then node i is reached again. There is also a way to express access times in terms of commute times:

$$H(i,j) = \frac{1}{2}\left(\kappa(i,j) + \sum_u \frac{d(u)}{2|E|}[\kappa(u,j) - \kappa(u,i)]\right)$$

We are now going to restate Theorem 3.1 and Corollary 3.2 from [39].

Theorem 1. *For a graph G with adjacency matrix A , let $0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_n$ where λ_i are the eigenvalues of the laplacian $L = I - D^{-1/2}AD^{-1/2}$, ϕ_i corresponding eigenvector and $\phi_{ij} = \sqrt{\phi(i)d(j)}$, and $d(u)$ the degree of the node u. Then we have the access time*

between two nodes u and v:

$$H(u,v) = 2|E| \sum_{l>0} \frac{1}{\lambda_l} \left(\frac{\phi_{lv}^2}{\sqrt{d(v)}} - \frac{\phi_{lu}\phi_{lv}}{\sqrt{d(u)d(v)}} \right)$$

and the commute time between two nodes u and v:

$$\kappa(u,v) = 2|E| \sum_{l>0} \frac{1}{\lambda_l} \left(\frac{\phi_{lv}}{\sqrt{d(v)}} - \frac{\phi_{lu}}{\sqrt{d(u)}} \right)^2$$

Remark 1. *In [39] they define commute times and access times spectrally but for normalized adjacency $N = D^{-1/2}AD^{-1/2}$. Since in our case we compute the symmetrical graph Laplacian look at $L = I - N$ we thought necessary to underline the correspondence between the eigenvalues. For N normalized adjacency let $\lambda_0 \geq \lambda_1 \geq ... \geq \lambda_n$ be its eigenvalues. Then for $L = I - N$ we have $\lambda_i^* = 1 - \lambda_i$ to be its eigenvalues. We reformulated the results from [39] to our settings which are $L = I - N$.*

Definition 3. *(Effective Resistance $\mathcal{E}$) Let a graph G with adjacency matrix A. The effective resistance $\mathcal{E}$ between two nodes u and v is:*

$$\mathcal{E}(u,v) = \frac{\kappa(u,v)}{2|E|} = \frac{\kappa(u,v)}{vol(G)}$$

These three metrics—Effective Resistance, Commute Time, and Access Time—play a crucial role in minimizing bottlenecks within graphs and facilitating the creation of rewirings that reduce commute time. However, computing these metrics often requires obtaining all eigenvalues and eigenvectors of the graph Laplacian, which can be computationally expensive. To overcome this challenge, we introduce a set of bounds that restrict the computations solely to the first eigenvalue, λ_1. By focusing on this single eigenvalue, we can significantly reduce the computational complexity while still obtaining valuable insights and making informed decisions regarding the rewiring process.

Remark 2. *Now we know that $\frac{1}{2} \leq \frac{1}{\lambda_k} \leq \frac{1}{\lambda_1}$ by [13]. By using this and the Cauchy inequality, we can arrive at the following result, proven by [39] in Corollary 3.3.*

Corollary 1. *We can bound the commute time κ*

$$|E| \left(\frac{1}{d(u)} + \frac{1}{d(v)} \right) \leq \kappa(u,v) \leq \frac{2|E|}{\lambda_1} \left(\frac{1}{d(u)} + \frac{1}{d(v)} \right).$$

Corollary 1 holds significant importance as it provides an upper bound for the commute time in terms of the inverse of λ_1. λ_1 refers to the first strictly positive eigenvalue of the Laplacian matrix $L = I - D^{-1/2}AD^{-1/2}$ and is commonly known as the *spectral gap*.

The Cheeger Constant Another metric closely associated with graph bottlenecks is the Cheeger constant [18]. The Cheeger constant serves as an essential measure, reflecting the graph's bottleneck properties and its connectivity characteristics.

Definition 4. *The Cheeger Constant represents the energy required to disconnect G into two communities, and it has the following formula:*

$$h_{Cheeg} = min_{U \subset V} \frac{|u, v \in E : u \in U, v \in V \setminus U|}{min(vol(U), vol(V \setminus U))}$$

Remark 3. *(Complexity of computing Cheeger Constant) In Definition 4 it can be noticed perhaps that we are taking all subsets U contained within a set V which has $\mathcal{O}(2^n)$ complexity. Nevertheless, as an example, if computing the Cheeger constant for a graph with 20 nodes takes one minute, then computing the Cheeger constant for a graph with 40 nodes takes around 2 years and so on. As our intent is to study graphs that require long-range interaction, using the Cheeger constant in the loss is unreasonable.*

Remark 4. *The Cheeger inequality which is $2h_{Cheeg} \geq \lambda_1 > \frac{h^2_{Cheeg}}{2}$, gives as a way to appeoximate the Cheeger constant with λ_1. Since computing the Cheeger constant is very costly, we use Cheeger inequality to approximate its value by $\lambda_1 \approx h_{Cheeg}$.*

We now have built up the theory that motivates the importance and expressivity of λ_1. Since we will use λ_1 in the loss of our Variational Autoencoder, moving forward underlying all these connections with Commute Times, and the Cheeger constant is very important.

Chapter 4

A Variational Approach

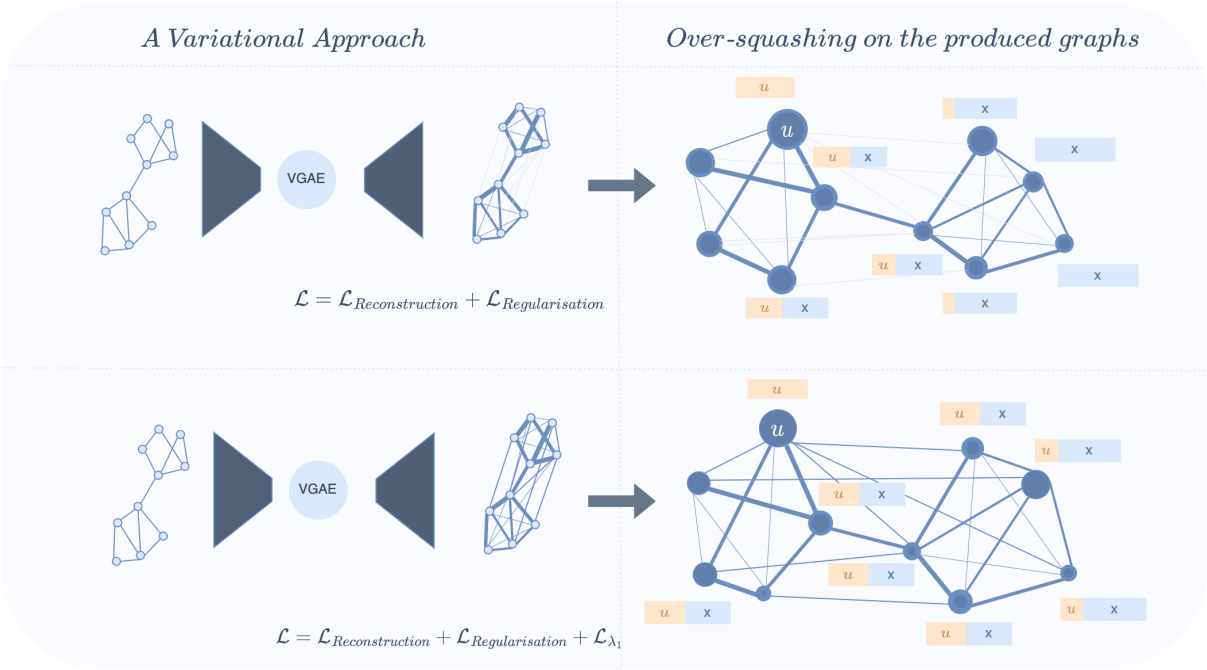

Figure 4.1: Variational approach summarisation. By including λ_1 in the loss function we will generate graphs that are less bottlenecked so that have better information flow. Pictorial representation of u influence propagating from one cluster to another, with x representing influence coming from other nodes.

In this section, we study a variational approach to rewiring. We delve into our methodology and then present our experiments and findings from this approach. In section 4.1, we present the general methodology of this approach. In section 4.2, we explore variational techniques to generate graphs. In section 4.3, we investigate the addition of λ_1 in the loss function. Finally, in section 4.4, we evaluate the rewiring graphs obtained in a downstream task, focusing on both peptides and ZINC datasets.

4.1 Methodology

We propose extending the VGAE loss function by introducing a new term alongside the reconstruction and regularization terms. We are motivated by the expressive power of the first positive non-zero eigenvalue of the Laplacian, which serves as a bound for metrics such as Commute time and the Cheeger constant. These metrics have been used in the rewiring community to assess graph bottlenecks [18, 2, 3].

$$\mathcal{L}_{\text{VGAE}} = E_{q(Z|X,A)}[\ln p(A|Z)] - \text{KL}[q(Z|X,A)||p(Z)] \tag{4.1}$$

We are interested in investigating the impact of introducing a parameter λ_1 into our loss function. The modified loss function is given by:

$$\mathcal{L}_{\lambda_1-\text{VGAE}} = E_{q(Z|X,A)}[\ln p(A|Z)] - \text{KL}[q(Z|X,A)||p(Z)] + \alpha\lambda_1(p(A|Z)) \tag{4.2}$$

To simplify notation, we denote $\lambda_1(p(A|Z))$ as $\lambda_1(A^*)$, where $A^* = p(A|Z)$ represents our rewiring graphs. It's worth noting that VGAE is a variational technique, generating probabilistic graphs where each edge's weight represents its probability of existence. The first positive eigenvalue of a graph can be computed even on weighted graphs with edge weights between 0 and 1, we can see an example of such computation in Figure 3.6. Our learning pipeline, illustrated in Figure 4.2, consists of two steps.

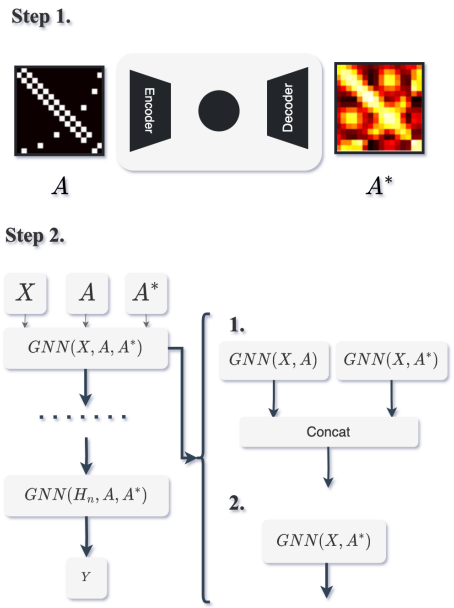

Figure 4.2: Variational pipeline in two steps. **Step 1.** Fully train the VGAE. **Step 2.** Include A^* in supervised task.

1. We fully train the VGAE to generate the rewired probabilistic matrices. We train the model for various values for α, this way we will be able to observe variations in the reconstructed adjacency matrices (from bottlenecked, to less bottlenecked).

2. We incorporate the newly generated adjacency matrix into a downstream task to empirically demonstrate its usefulness. There are many options in how we can build the layers but we will take into consideration only two: in parallel (number 1 in Figure 4.2), and without initial adjacency (number 3 in Figure 4.2).

We continue with our study on how we adapted VGAE to work in inductive tasks (that involve more graphs and regression over these) and a study on expressivity.

4.2 VGAE exploration

VGAE's initial design was meant for transductive tasks (i.e. edge prediction over one graph). Due to the relative lack of popularity and exploration of Variational Graph Autoencoders (VGAEs) in the context of inductive tasks (involving more graphs, and a graph-related task like classification), it becomes crucial to delve into different architectural variations and comprehensively understand their behaviour across diverse architectures. This entails investigating the performance of VGAEs under various training, testing, and validation splitting methods, as well as exploring different architectures.

By conducting such extensive explorations, we aim to gain insights into the efficacy and limitations of VGAEs in tackling inductive tasks with multiple graphs. This comprehensive analysis will enable us to better understand the behaviour of VGAEs in diverse scenarios, thereby facilitating the development of more robust and effective models for handling complex graph-based datasets.

All experiments in this section have been tried on ZINC. For Encoders, we have used various layer types (GCN, GIN, PNA), with various depths. We have tried two methods for dataset splitting. For the Test, and Validation Errors we compute the Average Precision (AP), just like it was done in [35]. As a prior, we have used a Gaussian just like the authors of [35]

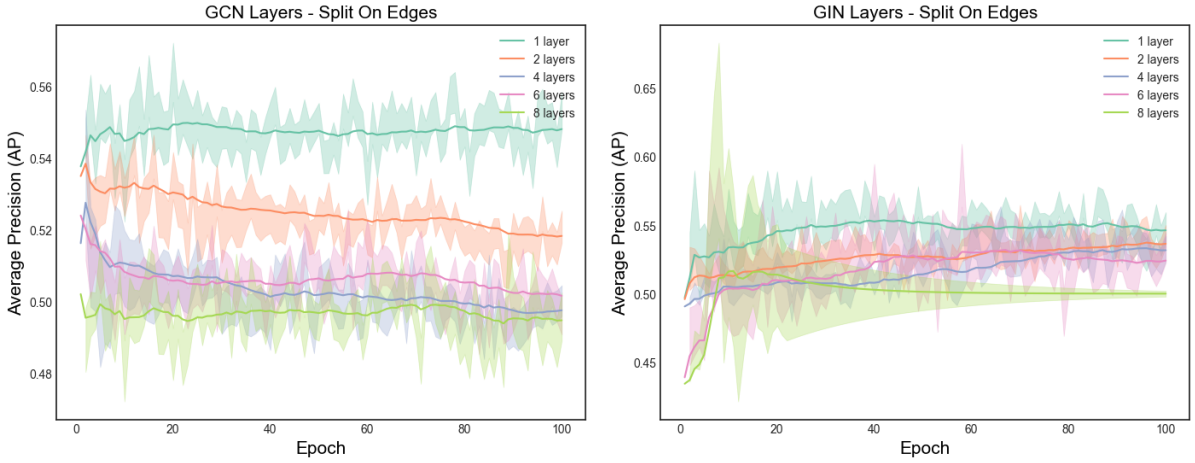

Figure 4.3: The model is trained on the ZINC dataset, but the data splitting is performed on the edges rather than on the individual graphs. The encoders are either GCN, or GIN based, and their performance is analyzed at various depths. The plots display the Average Precision (AP) scores over the test set. It can be observed that the test metric remains around 0.5, with no significant increase. A conclusion would be that edge splitting doesn't work in inductive tasks.

.

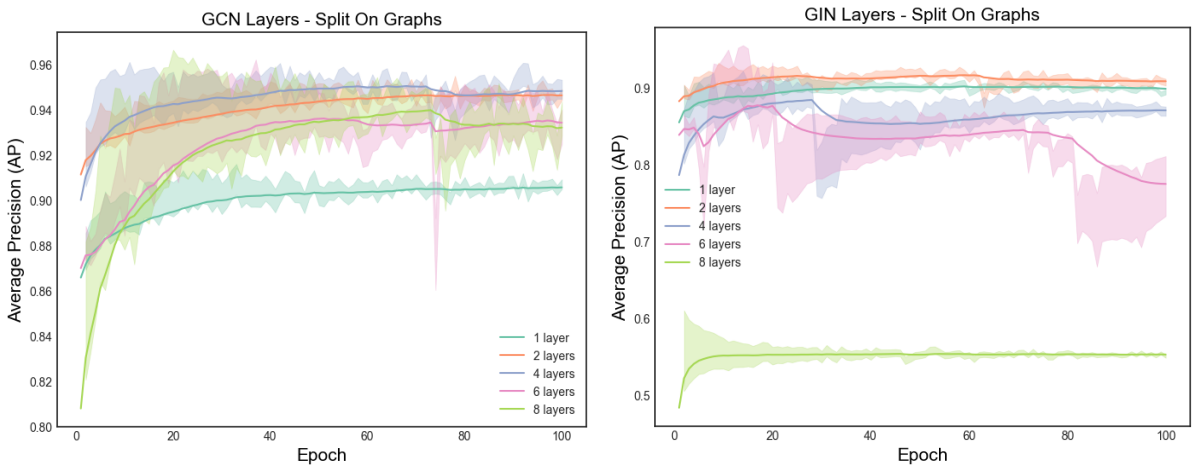

Figure 4.4: Training model on ZINC but splitting of the data takes place on the graphs. The model is now trying to learn how to rebuild full graphs. We can see that for both models our metric over the test set increases, meaning that the model learns. The shading is the moving average and standard deviation over a window of 10 Epochs.

4.2.1 Dataset splitting

VGAE [35] has originally been designed for one graph link prediction tasks on datasets such as Cora, Pubmed etc. A later version called GraphVAE [44], was designed for molecules autoencoding, but unfortunately, we weren't able to find any working codebase that had this model working, so in this section, we discuss how we switched the VGAE from solving transductive (one graph) tasks to inductive (multiple graphs) tasks.

In the case of transductive learning, the graph is split into training, testing and validation sets of edges. In the training phase, the model sees only the edges belonging to the training set, and it attempts reconstruction. In the validation and testing phase, the model sees the set of training edges and attempts a reconstruction of training edges, plus testing or validation. For our case, this approach is not very sensible because in doing so we break the molecule, a fully connected graph into smaller disconnected graphs. We have applied this approach to our case and proved empirically that it doesn't work. As we can see in Figure 4.3, where we plot the Average Precision on the validation set, over epochs our metric doesn't improve, which means this way of splitting the data doesn't help the model learn.

However, a more effective approach is to split the data based on the graphs themselves. By training the VGAE to reconstruct complete molecules and evaluating its performance on full graphs during the validation and testing phases, we can assess the quality of the generated graphs more accurately. This approach aligns well with our goal of introducing a third term in the loss function to evaluate the proposed graphs, specifically the value of the first positive Laplacian. As shown in Figure 4.4, the evaluation metrics on the validation dataset consistently improve during the training process, indicating that the model is learning and making progress.

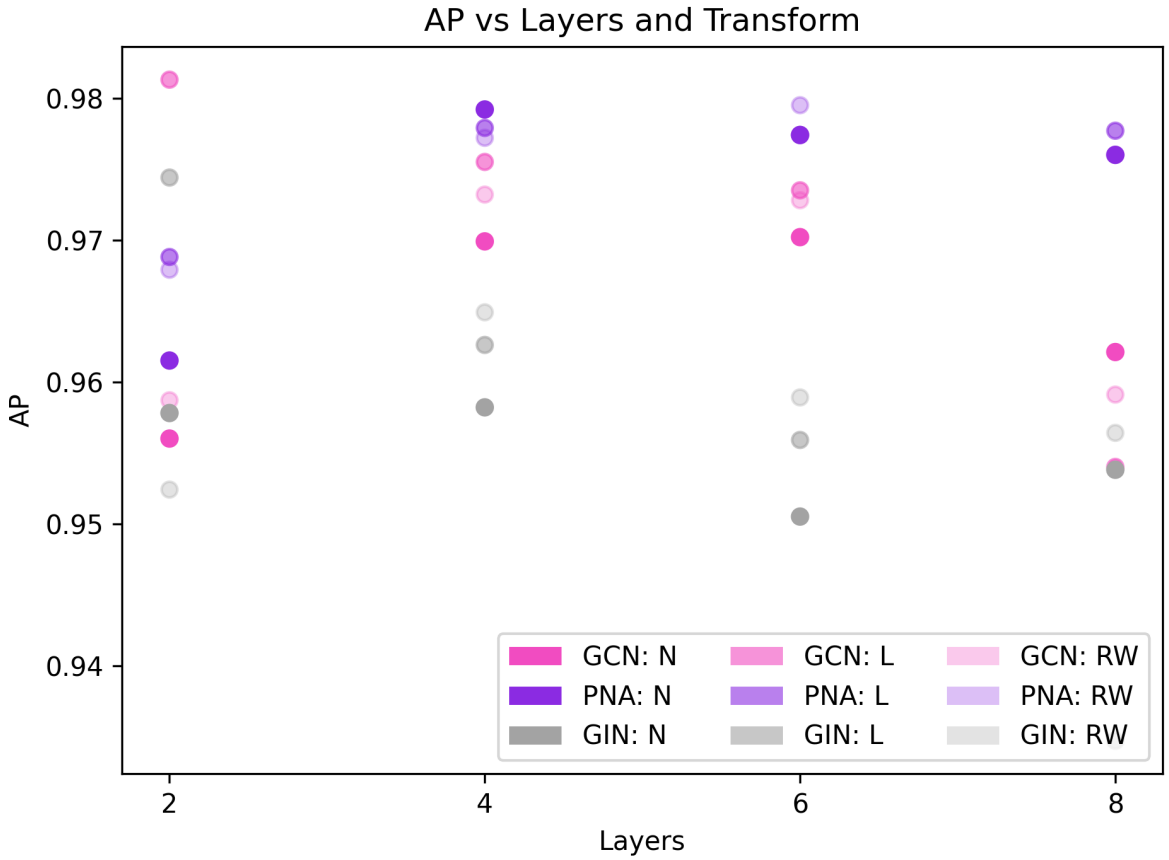

Figure 4.5: In this figure, we compare the expressivity of various layer depths and transformations, as well as the use of positional encoders. The three types of positional encodings considered are Laplacian (L), Random Walk (RW), and No positional encoding (N). Based on the results, it is evident that VGAE can serve as an effective framework for understanding layer expressivity. Notably, the comparison shows a clear separation: $PNA > GCN > GIN$.

Something else to notice in Figure 4.4 is the difference in performance between the two architectures. The GCN-based layers achieve accuracies around 94% in most cases with GCN with 1 layer being the outlier obtaining at most 90% average precision. The GIN-based layers struggle to hit 90% average precision, and also for 8 layers perform very badly. We are going to discuss in the next section general expressivity for VGAEs, because as we see in the two figures the architecture choice matters a lot for the deconstruction error.

4.2.2 Expressive power of VGAE

In an ideal scenario, our Variational Graph Autoencoder (VGAE) would possess the ability to accurately decode a graph, resulting in improved rewiring when incorporating λ_1 in the loss function. However, the reality is that the expressive power of the Variational Graph Autoencoder (VGAE) relies on the capabilities of its encoder and decoder. In this case, the decoder is entirely deterministic, just being an inner product between the hidden representation z and its transpose z^T. It is well-known that Graph Neural

Networks (GNNs) often face limitations, particularly in distinguishing cycles [49], which consequently affects their ability to decode graphs effectively.

We believe that variational autoencoders offer an empirical approach to gauge the expressivity of newly proposed graph layers. If a new layer exhibits superior performance in representing a graph within a lower-dimensional hidden space, particularly in terms of accurately reconstructing the original adjacency matrix, then it can be considered more expressive compared to previously proposed layers.

In the depicted Figure 4.5, we have conducted a comparison of different model types (GCN, PNA, and GIN) on the ZINC dataset. The evaluation involved exploring various transformations applied to the input feature vector x, including positional encodings such as No Positional Encoding, Random Walk Positional Encoding, and Laplacian Positional Encoding. These experiments were conducted across different layer depths.

Consistent with our previous observations, it is evident that layers known to possess higher expressive power tend to yield better performance. In particular, the PNA model exhibits superior results across various layer depths and transformations. Notably, for a depth of 2, the inclusion of positional encoding proves to be highly beneficial, leading to improved performance.

In Figure 4.6, we observe that the graphs generated by the PNA model tend to exhibit better connectivity compared to those generated by the GCN and GIN models. The GCN and GIN models often produce disconnected graphs, whereas the PNA model demonstrates a better ability to recognize and preserve cycles within the reconstructed graphs. This distinction becomes apparent when comparing the reconstructed graphs to their original counterparts. It is important to note that these observations are based on a limited subset of two graphs, but this behaviour tends to persist across a larger number of reconstructions.

4.3 Adding $\mathcal{L}_{\lambda_1}$ to the loss function

In this section, we investigate the impact of adding $\alpha\lambda_1(A^*)$ to the VGAE loss function. We begin by confirming that incorporating this term in the loss function leads to variations in the λ_1 values of the generated graphs. Subsequently, we conduct a detailed analysis of the generated adjacency matrices and explore the effect of applying a threshold over the generated matrices. Within this section, we conduct experiments using the encoders that achieved the best reconstruction results in the previous case. Additionally, we examine the performance of VGAE on a new dataset consisting of peptides, which are larger molecules.

As seen in Figure 4.6 and Figure 4.5 the choice of the encoder architecture matters a lot. For ZINC our encoder will be a PNA with a 4-layers based encoder since it obtains first-in-class results, and for Peptides it will be a 4-layered GCN because they obtained

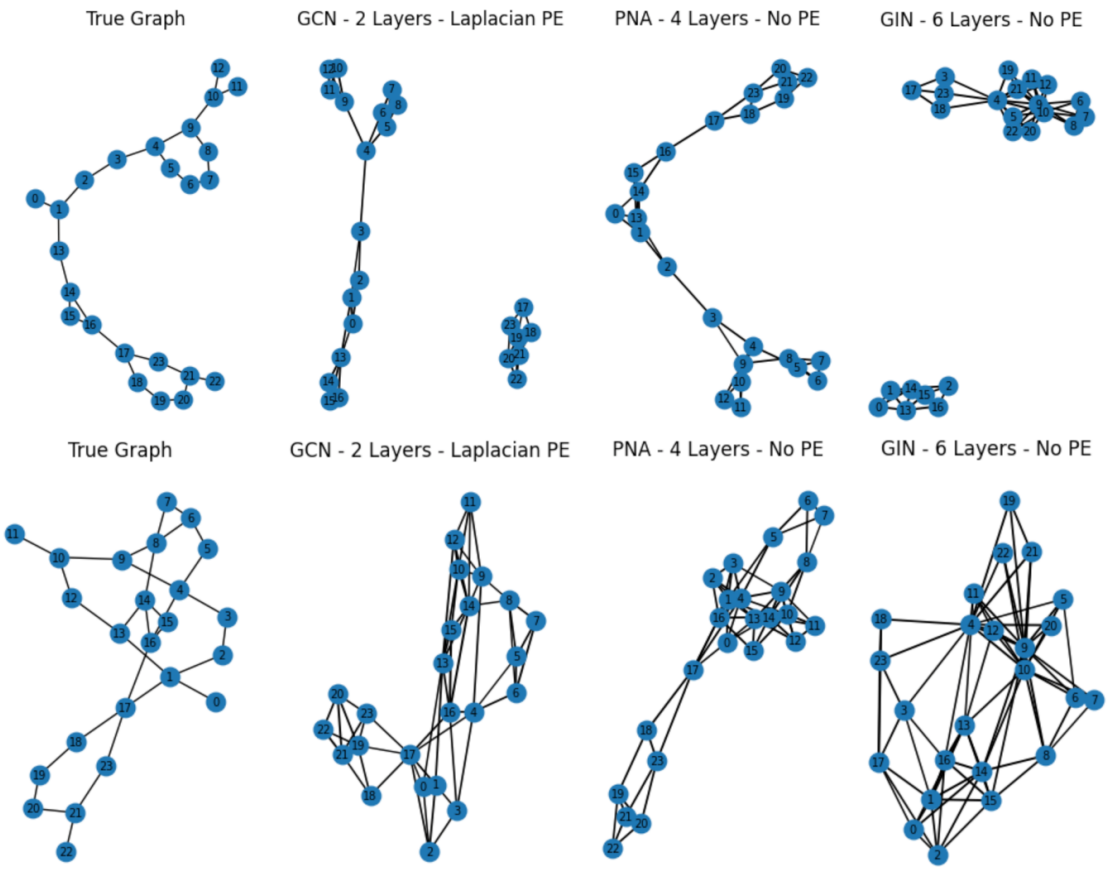

Figure 4.6: On the left the original graph and on the right reconstructed versions of the models described in this section. These graphs come from probabilistic adjacency matrices. The same type of processing which was a combination of thresholding ($p_{edge} > 0.5$) plus the use of k-nearest neighbourhoods (for $k = 4$), and also a limit of the maximum number of edges that can be added $= \frac{3}{2}|E|$ added on how many edges can be added has been chosen. As it can be seen the generated graphs are quite far away from the true ones.

the best reconstruction results over test sets. For ZINC we picked from the list present in Figure 4.5. For Peptides, we tested GIN and GCN-based layers with depths varying from 2 to 8, with 4-layer GCN obtaining the best average precision. We added no positional encoding.

4.3.1 Validating model

Up to this point, the loss function has been defined as follows:

$$\mathcal{L}_{\text{VGAE}} = E_{q(Z|X,A)}[\ln p(A|Z)] - \text{KL}[q(Z|X,A)||p(Z)] \tag{4.3}$$

We are now interested in investigating the effects of introducing a parameter λ_1 into our loss function. The modified loss function can be expressed as:

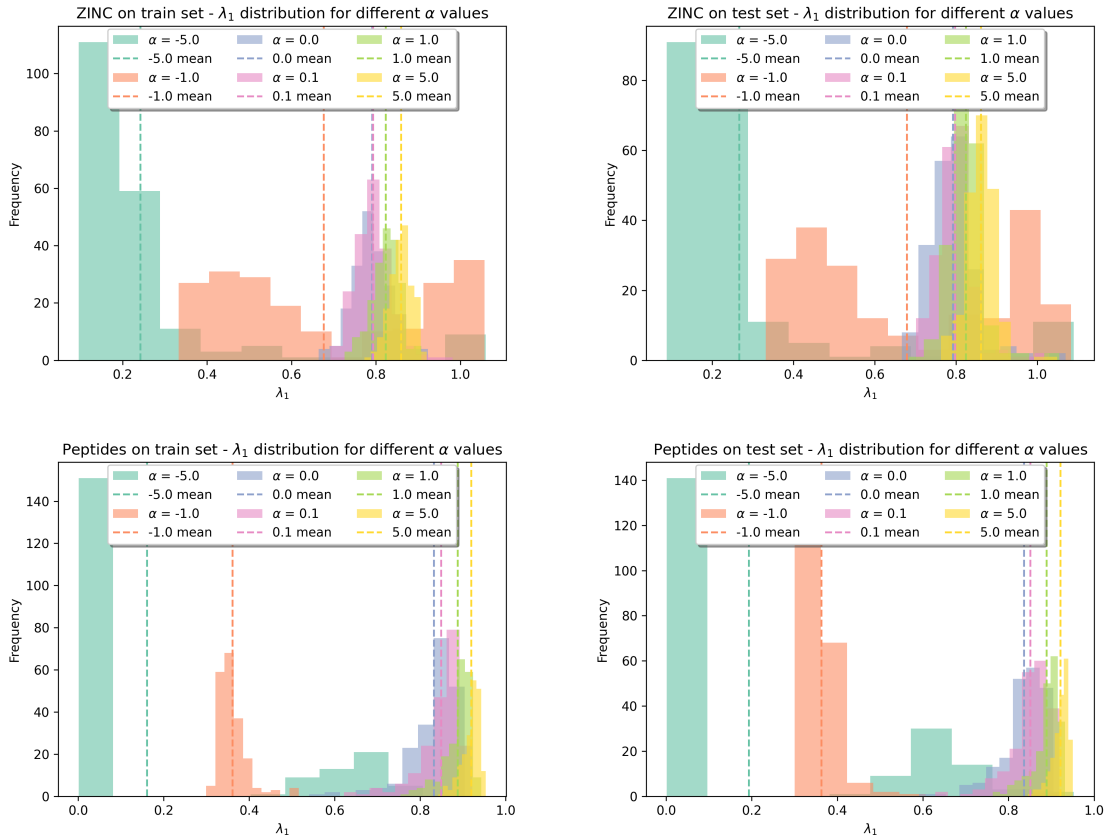

Figure 4.7: VGAE trained with the loss function from Equation 4.2, we have checked the λ_1 distributions of the generated graphs, for both training and testing sets. For ZINC we have used a 4-layered PNA-based encoder, and for Peptides 4 layered GCN has been chosen. As we can see in the figure the distributions $lambda_1$, as we increase the values of α in our loss the graphs get shifted to the right, which means less bottlenecked graphs will be generated.

$$\mathcal{L}_{\lambda_1-\text{VGAE}} = E_{q(Z|X,A)}[\ln p(A|Z)] - \text{KL}[q(Z|X,A)||p(Z)] + \alpha\lambda_1(p(A|Z)) \tag{4.4}$$

We would like to examine how variations in the value of the parameter α impact the reconstructed graph. Specifically, we want to understand the consequences of both increasing and decreasing the value of α.

As depicted in Figure 4.7, it is evident that as we increase the value of α, the model exhibits the ability to generate less clustered graphs, because the distributions of the generated graphs shift to the right. The figures also provide a reassuring observation: for graphs that were not encountered during the model's training, the order is maintained. To validate our hypothesis, we further evaluated the effectiveness of the modified loss function on a different dataset, namely peptides, which is known in the literature as a dataset that requires long-range interactions.

4.3.2 Reconstruction error over test dataset

In this section, we analyze the impact of changing the value of $\alpha \in \{-10, -5, -1, 0, 0.1, 1, 5\}$ in the loss function on both ZINC and Peptides. As we can see in Figure 4.8 for positive values of α the reconstruction of the graph is not disrupted, in some cases for $\alpha \in \{0.1, 1\}$, we actually get even better values for the average precision. We think this happens because the encoders have the tendency to form strong communities (as we can see in Figure 4.6) as they are not expressive enough and a positive coefficient for λ_1 encourages the model to generate less bottlenecked graphs, which are closer to the initial graph. For negative values of α the model is encouraged to form strong communities, so the very bottlenecked structure, which is not very commune in molecules so the Average Precision is hurt.

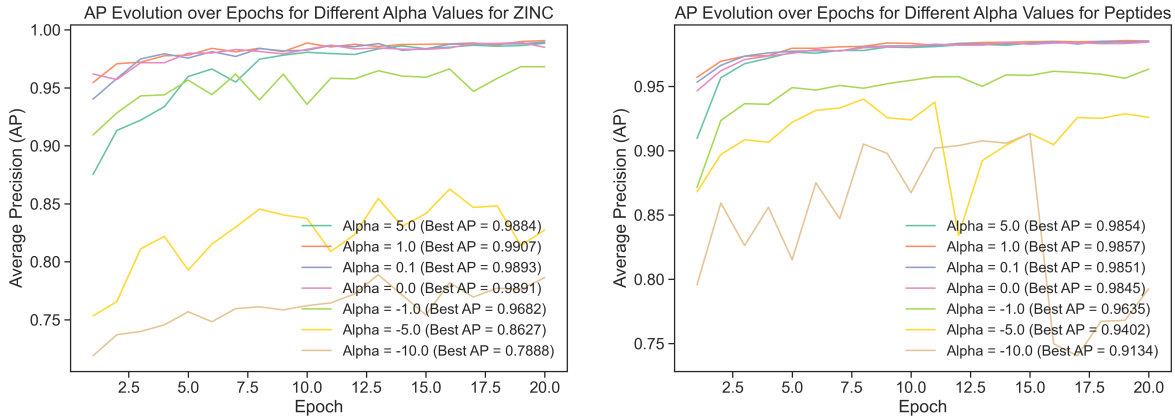

Figure 4.8: Average precision for encoders trained on values of α applied on ZINC and Peptides. We see as the coefficient of λ_1, α, increases the reconstruction metric increases as well. For $\alpha = 1$ the reconstruction metric is even better than for $\alpha = 0$, which means adding λ_1 to the loss helps.

4.3.3 Generated graph processing

As previously mentioned VGAE generates probabilistic adjacency matrices, which means that the graph is fully connected with each edge having weights between 0, and 1, numbers describing the probability they exist. We can observe some for three data points coming from the test dataset from both ZINC and Peptides in Figures 4.9 4.10, for various α values. In light colours, we have edges, with high probability, and in dark colours edges with lower probability.

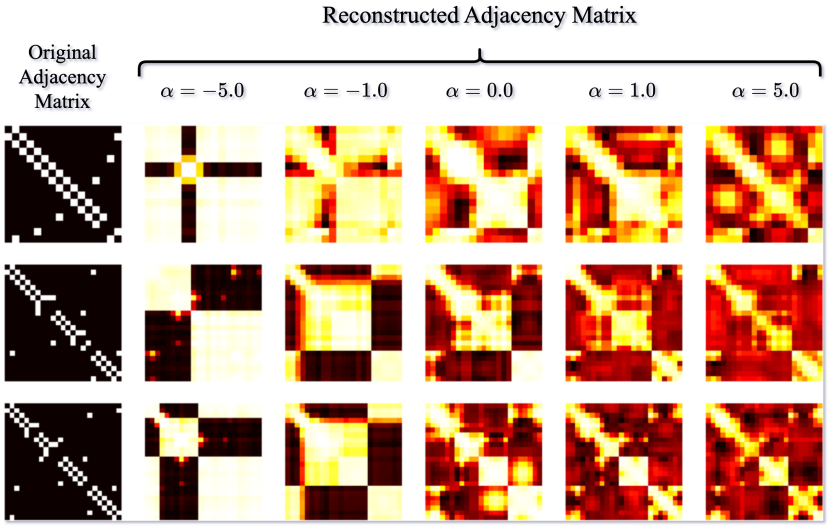

Figure 4.9: Reconstructed Adjacency matrices for various alpha values on ZINC. For negative α the graph is bottleneck, strong communities united by few edges. For positive values, the landscape start be similar to the initial adjacency matrices.

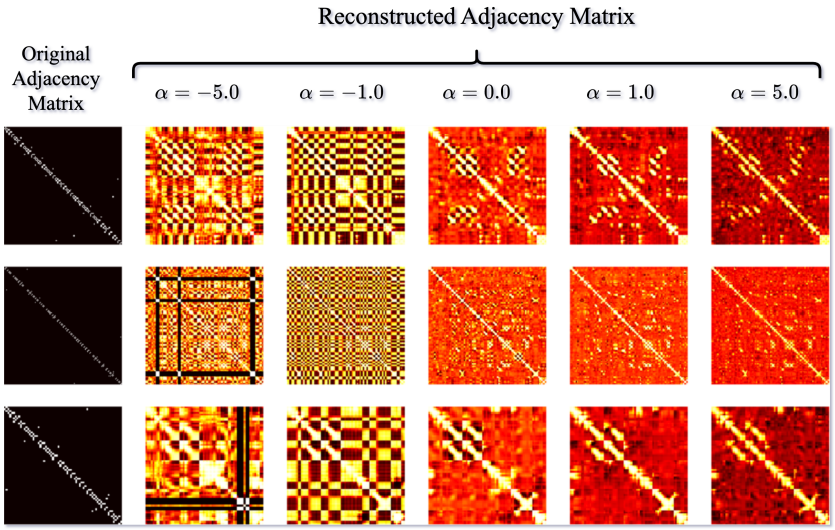

Figure 4.10: Reconstructed Adjacency matrices for various α values on Peptides. For $\alpha = -5.0$ we can observe the bottlenecks (the black stripes) but the communities are not as strong as they were for ZINC.

For ZINC, In Figure 4.9 row 2 for $\alpha = -5.0$ the bottleneck can be easily seen, we can notice two strongly connected communities, connected only by what looks like 4 weak other edges. In the Peptides case, in Figure 4.10 same pattern can be recognized for negative *alpha* values. In the case of peptides, in Figure 4.10 row 3, we can observe 4 relatively strongly connected communities united by only 1 node which is connected to everything. The bottleneck in this case is again easily visible. As α shifts to -1.0 we can notice that the communities tend to remain the same but the bottlenecks are alleviated, for example in Figure 4.10 on the 3rd row we can see that the bottleneck present at $\alpha = -5.0$ disappear at $\alpha = -1.0$. Something else very interesting is that even though the

models trained with different values of alpha are different moving from one value of alpha to the next same patterns can still be recognized in the background.

Utilizing this current adjacency matrix in our pipeline would essentially establish a graph transformer, where the edge weights within the adjacency matrix act as an attention mechanism. However, working with densely connected graphs entails significant computational expenses. Hence, in this section, we delve into potential approaches that can be explored to introduce sparsity to the graphs.

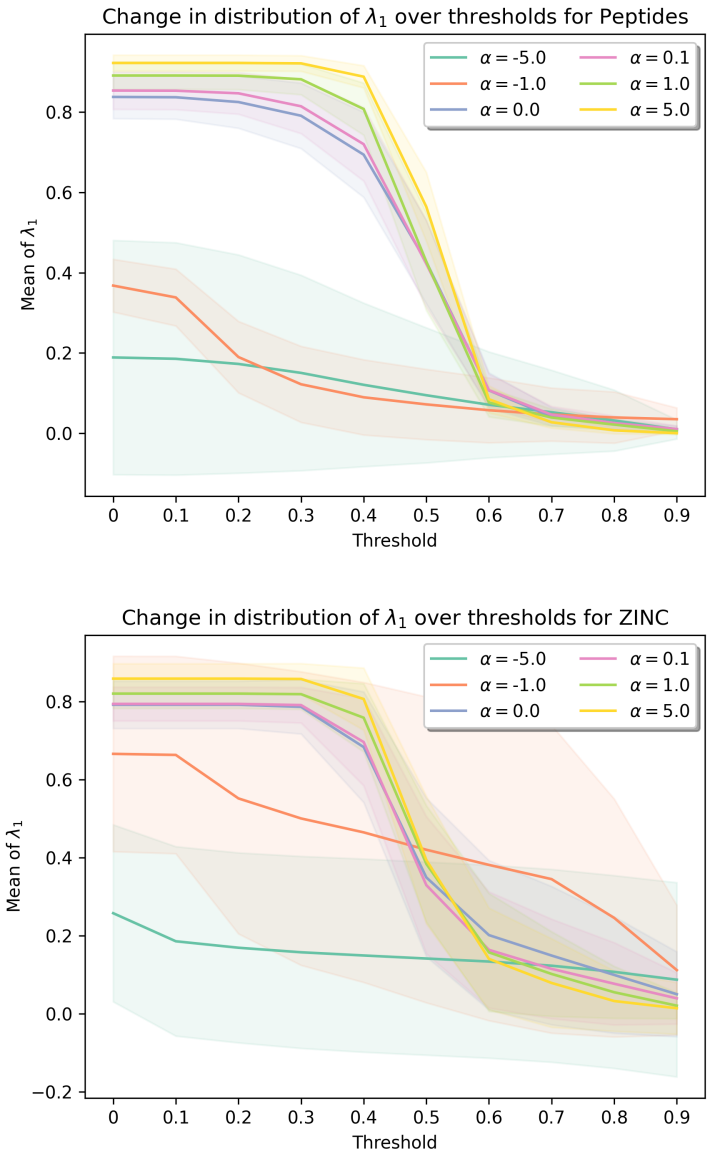

Figure 4.11: **Threshold can hurt:** We applied various thresholds over the generated adjacency matrices, and we learn that the connectivity (measured in λ_1) is very hurt for threshold values bigger than > 0.5. As it can be seen in the two figures at threshold $= 0.6$ the generated graphs with different values of α almost have the same mean in λ_1.

Threshold As depicted in Figure 4.11, when considering threshold values above 0.6, the connectivity of the graph is significantly compromised. Consequently, the coefficient

α in our loss function would no longer have any influence. Surprisingly, for thresholds above 0.7, the graphs generated by models with negative α exhibit better connectivity compared to the graphs generated by models with positive α.

This observation aligns with the fact that the VGAE models were specifically trained to enhance connectivity on the full adjacency matrix, rather than on processes involving thresholded adjacency matrices. For graphs generated by an encoder with positive α, the application of a threshold to the adjacency matrix leads to the removal of numerous edges, potentially causing the graph to become disconnected. Consequently, $\lambda_1(A^*[a_{i,j} > \text{threshold}])$ approaches zero, resulting in a drastic decrease in its value as the threshold is increased.

4.4 Evaluation in supervised tasks

In this section, we will evaluate the quality of the rewirings proposed by the VGAE. We will consider three sets of experimental architectures in which we introduce these rewirings within a supervised learning pipeline, on both Peptides and ZINC. The architecture can be viewed on the right-hand side, in Figure 4.12. At the end of this experimentation, we want to find out if A^* encodes useful information or helps the pipeline to achieve better results. We are going to use A^* coming from encoders that have been trained with values of α coming from $\{-5, -1, 0, 0.1, 1, 5\}$. As for the supervised models, the architectures we will choose are the following:

- For the first model we are going to choose a positional encoding of size 20, and chose option 1 from of GNN layer from Figure 4.12. For the GNN layer that learns on A initial adjacency matrix we are going to pick a layer of type GINE, and for the layer that learns on A^* we will pick a GCN layer. We are going to call this architecture $GNN(A, A^*)_{20}$

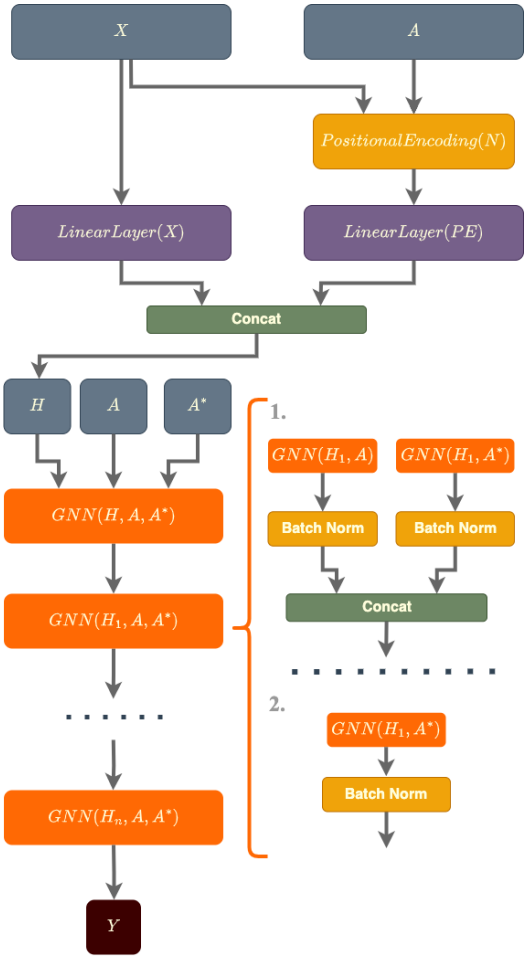

Figure 4.12: Learning pipeline - with 2 options for the $GNN(H, A, A^*)$. The first one concatenates information from a GNN that learns on the initial adjacency matrix A with a GNN that leans on A^*. The second one is a GNN that only learns on A^*.

- For the second model we are going to pick a positional encoder of size 20 and option 2 where the GNN layer will be again of type GCN. We will call this model $GNN(A^*)_{20}$.
- For the last architecture we are going to use a positional encoder layer of size 2, and option 2 with the GNN layer to be GCN. We will call this model $GNN(A^*)_2$.

We designed the tree architecture to reduce the information steadily to understand if the information from A^* is useful and also to study if bottlenecked A^* hurt the process (those with negative α). Each model type has GNN layers and has been trained for 150 Epoch. For Positional Encoding, we have used Random Walk. For batch size, we have picked 64 for ZINC and 20 for Peptides.

For both ZINC and Peptides-struct, the loss, validation and test error are calculated with Mean Absolute Error (MAE), which means smaller values are better.

4.4.1 General Results

Firstly, in Tables 4.2 and 4.1 we can observe a natural progression in terms of the results. As more and more useful information is removed' firstly the initial adjacency matrix A, and then the positional encoding size, both the loss and smallest validation and the corresponding test value, decrease.

Architecture	Alpha	Smallest Loss	Smallest Validation Error	Corresponding Test
$GNN(A, A^*)_{20}$	-10.0	0.4003	0.3199	0.3129
	-5.0	0.4054	0.3334	0.3201
	0.0	0.3907	0.4341	0.437
	1.0	0.3911	0.4081	0.3807
	5.0	0.3972	0.4166	0.4045
$GNN(A^*)_{20}$	-10.0	0.5095	0.4087	0.403
	-5.0	0.4962	0.3787	0.3963
	0.0	0.4933	0.3826	0.3676
	1.0	0.4944	0.3746	0.3803
	5.0	0.4966	0.3801	0.3401
$GNN(A^*)_2$	-10.0	0.5977	0.4971	0.5262
	-5.0	0.5865	0.4756	0.5123
	0.0	0.5941	0.5030	0.5332
	1.0	0.5894	0.4920	0.5142
	5.0	0.5976	0.4997	0.5229
$GNN(\mathbb{1})_2$	-	0.4503	0.5388	0.5853

Table 4.1: ZINC Results: In red signs over-fitting, and in best-results for a model type. $GNN(\mathbb{1})_2$ means that the model has been trained on a GNN where the edge weights were all 1.

ZINC Analysis In this paragraph, we will examine the results on the ZINC dataset, row by row, starting from $GNN(A, A^*)_{20}$, followed by $GNN(A^*)_{20}$, $GNN(A^*)_2$, and finally $GNN(\mathbb{1})_2$.

- Let's first examine the performance of $GNN(A, A^*)_{20}$. The best results are obtained when using $\alpha = -10$, which corresponds to an extremely bottlenecked landscape as depicted in Figure 4.9. However, for positive values of α, we observe signs of overfitting. Although these models achieve better Loss values, the Validation and corresponding Test errors are noticeably larger. Specifically, for $\alpha = -10$, the Test error is 0.31, while for $\alpha > 0$, it exceeds 0.38. We believe this occurs because the landscape for $\alpha = -10$ appears more regular, exhibiting four clusters of strongly connected communities linked by a small number of edges.

- Next, let's consider the experiments that exclude information from the initial adjacency matrix A and instead utilize positional encoding of size 20. In this case, the best test error, corresponding to the best validation error, is achieved with $\alpha = 5$. Interestingly, there are no indications of overfitting. However, the results obtained in this pipeline are inferior to those obtained when feeding the initial adjacency matrix to the model. It is also worth noting that the overfitting observed in the previous model $GNN(A, A^*)_{20}$ is likely due to the introduction of the initial adjacency matrix A into the pipeline. In the case of $GNN(A^*)_{20}$, the results seem to be better for positive α than in $GNN(A, A^*)_{20}$.

- Moving on to $GNN(A^*)_2$, we observe a significant increase in Loss by 0.1, which is quite substantial for the ZINC dataset. Here, the best performance is again achieved with a negative $\alpha = -5$.

- Finally, in the experiment involving $GNN(\mathbb{1})_2$, clear signs of overfitting are noticeable, with a small loss but significant validation and test errors.

These results are not surprising. For the ZINC dataset, models that effectively capture the structure of the initial graph usually achieve the best performance. In our case, we gradually removed elements containing information about the structure, such as A and the positional encoding was reduced from size 20 to 2. We believe that in the $GNN(A, A^*)_{20}$ row, the model attempted to learn the structure from A^*, resulting in overfitting to a structure that did not generalize well to the test set.

Peptides Analysis We will go through the rows of the Table 4.2, concerning models performance starting from $GNN(A, A^*)_{20}$, followed by $GNN(A^*)_{20}$, $GNN(A^*)_2$, and finally $GNN(\mathbb{1})_2$.

- Firstly for $GNN(A, A^*)_{20}$ it looks like the models are not influenced much by the change in the coefficient α.The best result is obtained for $\alpha = -5.0$, but considering how close the values from Loss, Validation and Test errors are to each other we cannot be sure it is actually performing better. Something very interesting to notice is that the values for Loss are smaller in $GNN(A, A^*)_{20}$ than in the case of $GNN(A^*)_{20}$, but the test and validations values are considerably bigger. I think this means some kind of over-fitting takes place on the initial adjacency matrix A.

Architecture	Alpha	Smallest Loss	Smallest Validation Error	Corresponding Test
$GNN(A, A^*)_{20}$	-10.0	0.2797	0.2830	0.2824
	-5.0	0.2776	0.2795	0.2819
	0.0	0.2785	0.2855	0.2877
	1.0	0.2784	0.2808	0.2832
	5.0	0.2776	0.2816	0.2833
$GNN(A^*)_{20}$	-10.0	0.2986	0.2693	0.2768
	-5.0	0.3080	0.3029	0.3134
	0.0	0.3022	0.2727	0.2781
	1.0	0.2969	0.2647	0.2695
	5.0	0.2929	0.2681	0.2755
$GNN(A^*)_2$	-10.0	0.3070	0.2869	0.2874
	-5.0	0.3101	0.2992	0.3084
	0.0	0.3248	0.3330	0.3422
	1.0	0.3166	0.2958	0.3062
	5.0	0.3040	0.2808	0.2859
$GNN(\mathbb{1})_2$	-	0.3041	0.3039	0.3153

Table 4.2: Results on Peptides. In green, we underlined best performing models.

- As mentioned previously it is worth noticing the Loss, Validation and Test Error discrepancy on $GNN(A, A^*)_{20}$, $GNN(A, A^*)_2$. Continuing, we observe that the best value is obtained for $\alpha = 1.0$. Which had the best reconstruction error if we remember Figure 4.8 B. It is worth noticing that the performance of this model beats the classical models' performance like GCN, GCNII, GINE, GatedGCN, GatedGCN+RWSE, which struggles to get results above 0.33 for Test MAE, and also comes very close to Transformer, and SAN Test MAE which is around $0.25 - 0.26$ [20]. We think this is remarkable because our networks don't use an attention mechanism they use just the landscapes we learned, from the VGAE.

- For $GNN(A^*)_2$ we notice a drop in performance, which means on peptides, the structure of the graph is still important. Something else to notice is the relative symmetry in results around $\alpha = 0.0$. A model that learned on $\alpha = -10$ got similar results to the model that learned on $\alpha = 5.0$, and similar symmetry for $\alpha = -5$ and $\alpha = 1$. This symmetry perhaps shows that strong bottlenecks don't hurt performance.

- Lastly, we have a model that had little information about the structure, which seemed to have to overfit, in comparison with $GNN(A^*)$.

Comparison between results on ZINC and Peptides The comparison of the results from the tables reveals distinct characteristics in how the two tasks respond to the input information. In the case of ZINC, it becomes evident that the inclusion of any form of structural information related to the initial graph is crucial for achieving favourable results. As we decrease the presence of this structural information, a noticeable decline in performance becomes apparent. On the other hand, for the Peptides task, while the structure remains important, the connectedness of the graph assumes greater significance,

not solely in terms of bottlenecks, but in relation to diameters. For instance, in Figure 4.10, strong bottlenecks can be observed for peptides. However, if the communities within the graph are well-connected, the information can reach nodes in adjacent communities within just two steps, as determined by the diameter, rather than the eigenvalue λ_1.

4.4.2 Results regarding threshold

Model	Th	Alpha	Loss	Val	Test
$GNN(A, A^*)_{20}$	0.0	-10	0.2797	0.2830	0.2824
		-5.0	0.2776	0.2795	0.2819
		0.0	0.2785	0.2855	0.2877
		1.0	0.2784	0.2808	0.2832
		5.0	0.2776	0.2816	0.2833
$GNN(A, A^*)_{20}$	0.4	-10.0	0.2798	0.2754	0.2714
		-5.0	0.2775	0.2808	0.2780
		0.0	0.2758	0.2873	0.2899
		1.0	0.2774	0.2902	0.2889
		5.0	0.2841	0.2593	0.2649

Table 4.3: Peptides with Threshold: We can see that applied threshold has improved to performance of the model $GNN(A, A^*)$, obtaining on test error a competitive results for $\alpha = 5.0$.

Conv	Th	Alpha	Loss	Val	Test
$GNN(A, A^*)_{20}$	0.0	-10.0	0.4003	0.3199	0.3129
		-5.0	0.4054	0.3334	0.3201
		0.0	0.3907	0.4341	0.4371
		1.0	0.3911	0.4081	0.3807
		5.0	0.3972	0.4166	0.4045
$GNN(A, A^*)_{20}$	0.4	-10.0	0.4049	0.3145	0.3159
		-5.0	0.4072	0.3457	0.3726
		0.0	0.3957	0.3286	0.3363
		1.0	0.3886	0.3313	0.3242
		5.0	0.3892	0.4708	0.4644

Table 4.4: ZINC With Threshold: For ZINC it does not look like the threshold made any major difference.

As depicted in Figure 4.11, using a threshold greater than 0.6 can significantly disrupt the graph connectivity in A^*. This prompted us to investigate the impact of a threshold of 0.4 on both Peptides and ZINC datasets. Tables 4.4 and 4.3 demonstrate that the threshold can have both positive and negative effects on performance.

In the case of ZINC, the threshold appears to be beneficial for α values of 0.0 and 1.0, but detrimental for $\alpha = 5$. On the other hand, for Peptides, employing a threshold seems to yield results that are comparable to the state-of-the-art range reported in [20]. However, we observed that the threshold's impact is somewhat unstable. Consequently, we propose that modifying the VGAE architecture to generate graphs with weights ranging from 0.6 to 1 would provide greater stability compared to processing the obtained graphs.

4.5 Final Experiment

The results from Table 4.2 are possibly noisy, and we wanted to learn if there is an order of efficiency over $GNN(A^*)$ depending on what was the value of the α the model was trained on (for example saying that less bottlenecked graph rewiring are doing better then very bottlenecked). To answer question this we run $GNN(A^*)_{20}$ for 5 seeds over $\alpha \in \{-10.0, 1.0, 5.0\}$. The results can be observed in Table 4.5.

In Table 4.5 we can see the following ordering. The graph rewirings that come from models that were trained with values of $(\alpha = -10)$ performed the best, followed by graphs coming from models that were trained on $(\alpha = 1)$, and the last $(\alpha = 5)$. This order is unnatural

Alpha	Loss		Validation Error		Test Error	
	Mean	Std	Mean	Std	Mean	Std
-10.0	0.298450	0.000870	0.269125	0.004998	0.271975	0.005632
1.0	0.300833	0.001761	0.272208	0.005251	0.276508	0.005336
5.0	0.306800	0.009302	0.281233	0.013731	0.284000	0.014297

Table 4.5: Mean and Standard Deviation of Loss, Validation Error, and Test Error for each Alpha

since, for ($\alpha = -10$) we get the most bottlenecked graph rewiring, and then for ($\alpha = 5$), the least. We can see that generally as α increases also the loss increases, which means that the models are actually able to learn less in the least bottleneck case than in the most bottleneck case. We think that the landscapes we generated are unstable and noisy, and it is hard for the supervised model to actually make sense out of them.

4.6 Variational Approach Conclusion

In conclusion, our exploration of the Variational Approach using the VGAE architecture and the incorporation of the spectral gap λ_1 in the loss function has provided valuable insights. We started with three questions: (1) Can such a model VGAE learn to build less bottlenecked graphs by placing $\mathcal{L}_{\lambda_1}$ in the loos? (2) is the generated graph helpful in a supervised setting? (3) does alleviating graph bottlenecks help in long-range supervised tasks?

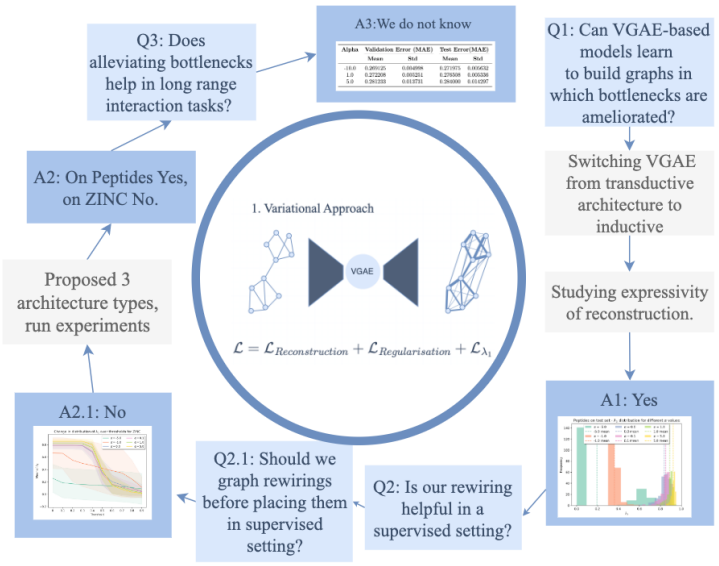

Figure 4.13: Variational Approach Conclusion Schema

On our way to answer these three questions, we first had to study how we can switch VGAE an architecture designed for transductive tasks to inductive tasks. We then study how different encoders, helped the architecture to decrease the deconstruction loss. We learned that the choice of layers, their, number, and the use of positional encoders were significantly influencing the reconstruction error. After choosing the best encoder types for each task, we finally got to answer the first question we had and place $\mathcal{L}_{\lambda_1}$ in the loss function. With Figures 4.7, we validated our model and answered the first question from our set of questions.

Following, we wanted to know if our graph rewirings actually help in long-interaction

types and we implemented 3 architecture sets on both ZINC and Peptides. For ZINC we saw no improvement when adding our rewirings to the pipeline, that was expected, however, because ZINC is a structural graph task rather than a long interaction one. For Peptides, we noticed improvement, in the sense that the model that has been fed solely the reconstructed adjacency performed better than the one which learned on the initial adjacency and then the one which learned on a fully connected with equal weights adjacency.

The last question we had was that if graph rewirings generated with negative α would perform worse than those generated with positive α. From Table 4.5 we learn that what is happening is the opposite of what we were expecting, meaning that the supervised model did not know how to interpret the non-bottlenecked landscape we fed into it.

Chapter 5

An Attentional Approach

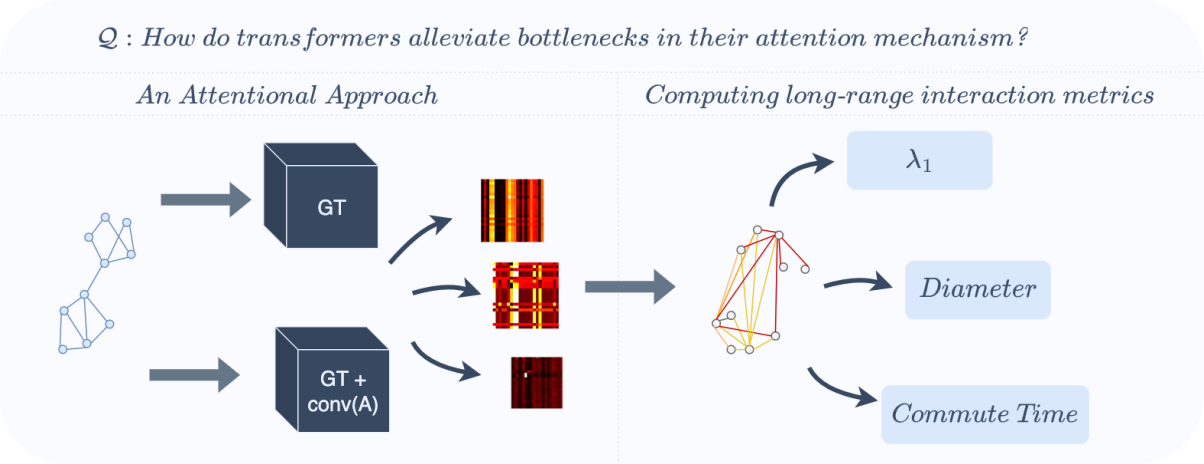

Figure 5.1: We will look at two architectures: a simple Graph Transformer (GT), and a Graph Transformer that will also embody graph convolutions on the initial adjacency matrix. We will extract the attention coefficients matrices at each layer for a population of graphs and analyze their long-range properties: λ_1, diameter, and Commute Times.

In this section, we will delve into an attention-based approach to address the problem of over-squashing. As discussed in the Background section (Section 3.4), transformers are capable of capturing long-range dependencies by connecting all nodes to each other [19]. Attention mechanisms play a crucial role in computing weights for node pairs, resulting in an attention coefficient matrix A. For each layer l we will get one attention head, such a matrix A^l can be computed, where $A^l[u, v]$ corresponds to the attention coefficient a_{uv} defined previously (Section 3.4).

Moving next, we will propose two Graph Transformer models, build upon GraphGPS [43], architectures which can be viewed in Figure 5.2. The first one will be a simple Graph Transformer which we will note as GT. And the second one will be a graph transformer that at one layer does message passing on the fully connected adjacency, and in parallel does message passing on the initial adjacency. The representations from both models will be concatenated, and pass through a linear layer, which will reduce the size of our choice

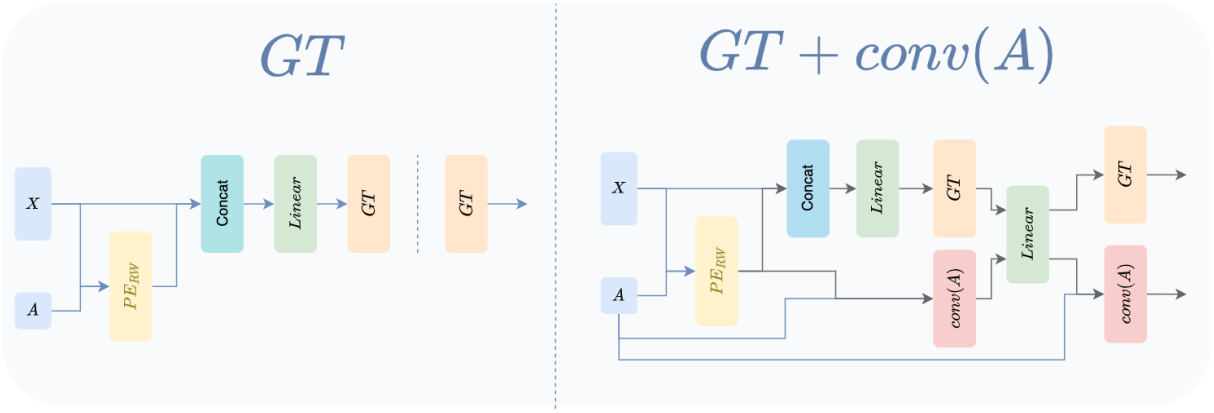

Figure 5.2: **GT** and **GT+conv(A)**: Picture of the two architectures. GT which does message passing on the fully connected adjacency matrix. $GT + conv(A)$ which does message passing both on the initial and on the fully connected adjacency matrix.

of hidden dimension which is 64. We will call this second model $GT + conv(A)$.

As Graph Transformers usually use positional encoding, we will use a Random Walk positional encoding of size 20. In the case of the ZINC dataset, we applied Embedding Layers for both edge attributes and node attributes, setting the size to 64. However, for the Peptides dataset, these layers were replaced with linear layers. The positional encoding also underwent a linear layer, resulting in a hidden dimension size of 64. In the case of $GT + conv(A)$, we considered one layer to be a combination of both the GT layer and $conv(A)$ layer, and the convolutional layer utilized the $GINE$ convolution [30]. Both models have 5 layers and one attention head at each layer.

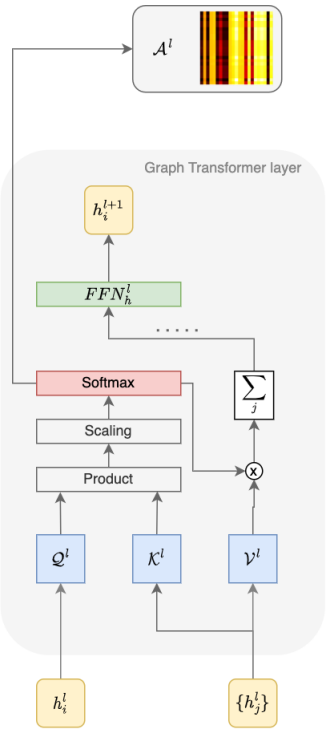

Figure 5.3: Graph Transformer: Forward-hook to obtain the attention matrix

After training both models on the Peptides and ZINC datasets for 300 epochs, we saved the weights of the models that achieved the best performance on the respective validation datasets. This ensured that the models were expected to perform well on unseen data during the testing phase. For the ZINC dataset, the $GT + conv(A)$ architecture achieved a test accuracy of 0.1605, while the simple GT architecture achieved a test accuracy of 0.1914. These results fall within the expected range for transformers, which typically yield values between 0.15 and 0.3 for the ZINC dataset [20]. For the Peptides dataset, the $GT + conv(A)$ architecture achieved a test accuracy of 0.26. The simple GT architecture after achieving a test accuracy of

0.2780. These values align with the expected performance range for transformers on the Peptides dataset, which typically range from 0.25 to 0.35 [22].

Overall, the obtained results demonstrate that both architectures performed well and achieved accuracies within the expected range for their respective datasets. Subsequently, we proceed to evaluate our test sets by passing one graph at a time through the architecture. By utilizing forward hooks (procedure depicted in Figure 5.3) we obtain the outputs of the attention layers. This allows us to compute the attention matrix for each graph, specifically at each layer. As our architecture consists of 5 layers of Graph Transformers (GT) with a single attention head, we obtain a set of 5 attention matrices for each graph.

5.1 Study over the landscape of coefficient attention matrices

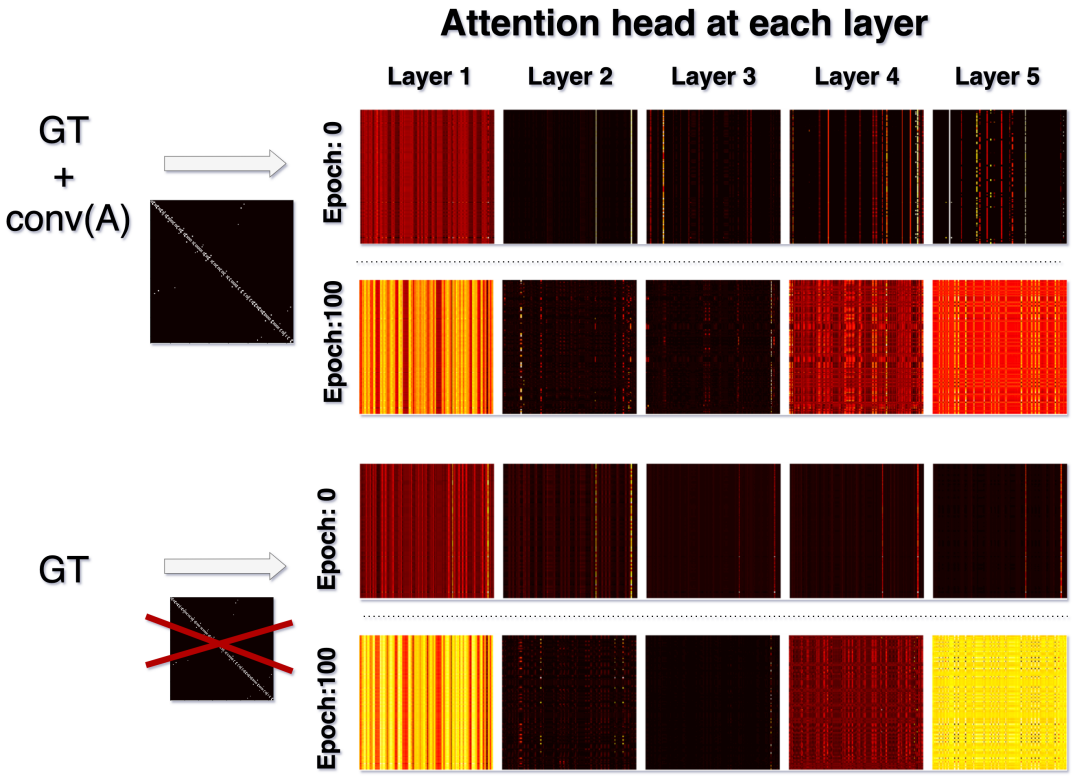

Figure 5.4: Heatmap over the attention learned over a graph coming from Peptides. The vertical stripes mean that some nodes are strongly connected with every other node in the graph. The first, and last layers seem to contain the most edges.

In Figures 5.4 and 5.5, we can explore the attention landscapes for graphs derived from both the Peptides and ZINC datasets using the simple GT and $GT + conv(A)$ models. Upon closer examination, several intriguing observations come to light.

In the case of the Peptides dataset, it is evident that the transformers tend to focus on what we refer to as *master nodes*. These master nodes are highly

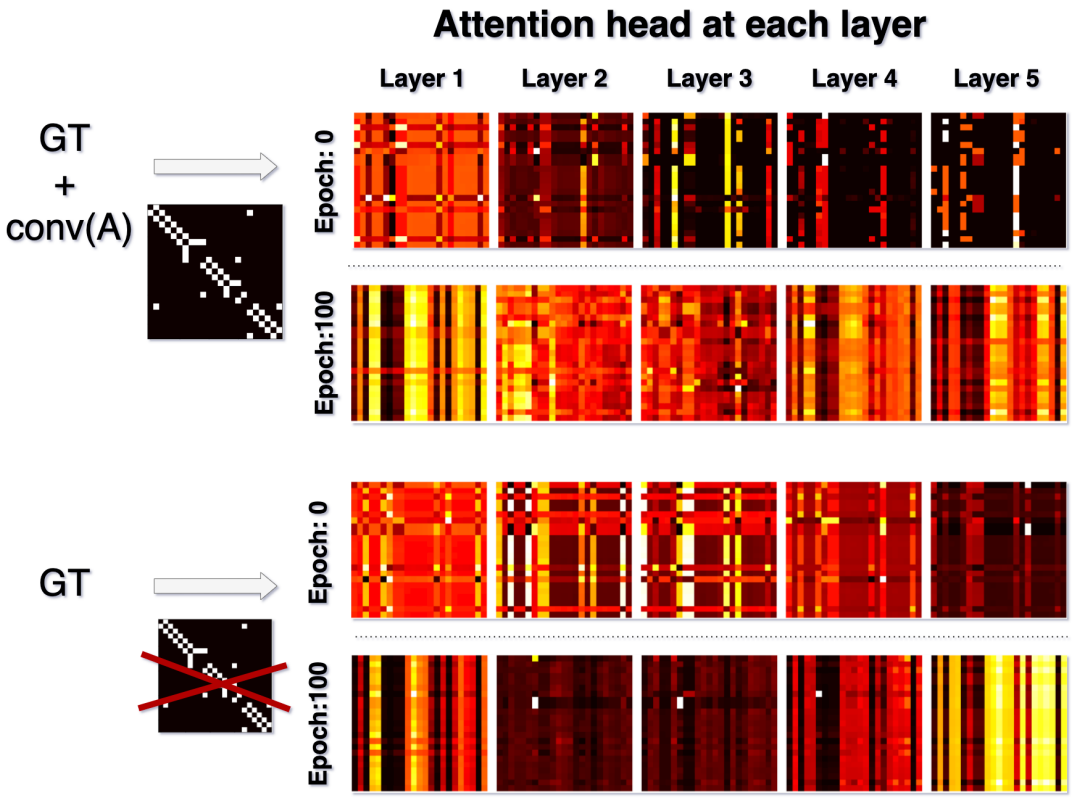

Figure 5.5: Heatmap over the attention learned over a graph coming from ZINC. The vertical and horizontal lines of different intensities can be interpreted as *master nodes*, nodes that are connected with every other node. there is a big difference between $GT + conv(A)$ and simple GT, at layers 2 to 4 where, for $GT + conv(A)$ master node structure dissapears.

connected and exhibit connections to every other node in the graph (the vertical stripes that can be viewed in Figure 5.4. Interestingly, there is no significant disparity observed in the attention landscapes between the GT simple and $GT + conv(A)$ models. Moreover, when examining the attention at the first layer, the primary difference lies in the intensity of the attention coefficients assigned to the nodes.

In contrast, the attention landscapes for the ZINC dataset display more noticeable discrepancies between the GT simple and $GT + conv(A)$ models. Specifically, the GT simple model appears to utilize the same attention coefficient filter across all depths, albeit with varying intensities. Conversely, the $GT + conv(A)$ model presents additional signals in the attention landscapes, particularly in Layer 2 and Layer 3, which exhibit distinct patterns compared to the other layers. This divergence in attention coefficients between the ZINC and Peptides datasets can be attributed to the nature of the ZINC task, which involves encoding graph structures. Hence, a proficient graph neural network (GNN) capable of understanding and capturing the underlying graph structure is expected to excel in the ZINC dataset. On the other hand, Peptides revolve around long-range interactions, and the attention landscapes reflect this unique characteristic.

5.2 Attention correlation to adjacency

We believe looking at some attention landscape has been useful, in the sense that it has been revealed to us that *master nodes* seem to be the mechanism that transformers mainly use in the attention mechanism. However, we were curious if the attention matrices were somehow correlated with the initial adjacency matrix. From Table 5.1, we learned that the correlation is very weak, and when the initial adjacency is added it becomes strictly negative.

In Table 5.1, we can see that for ZINC *GT* the correlations coefficient between the attention landscapes and the initial adjacency matrix was higher than in the case of $GT + conv(A)$, which means the model tries to learn the structure of the initial adjacency matrix in the attention coefficient matrices, especially at layer 2, and 3. For Peptides, this doesn't seem to be the case. As mentioned before ZINC is a task in which models that understand graph structure perform best. For example, CIN networks from [9], which build a structure around the cycles of the molecules, and enforce message passing between cycles currently obtain state-of-the-art results on this task.

For both cases, we must notice that as we add the $conv(A)$, the correlation between the adjacency matrices and the attention coefficient matrices becomes negative, which means the filters become less interested in discovering structure.

Table 5.1: Pearson Correlation between the attention coefficient matrix and initial adjacency matrix at each layer, for two datasets over two models. For ZINC *GT* we can notice there exists some correlation between the initial adjacency and attention coefficient, but for the rest cases, it is either very small or negative.

Dataset	Model	Correlation between Attention and initial A				
		1	2	3	4	5
ZINC	**GT**	-0.0060	0.0853	0.0787	0.0224	0.0191
	GT + Conv(A)	-0.0090	-0.0536	-0.0374	-0.0049	-0.0273
Peptides	**GT**	-0.0236	-0.0101	-0.0018	-0.0087	0.0188
	GT + Conv(A)	0.0004	-0.0008	-0.0117	-0.0123	-0.0242

5.3 Long-Range Interaction Study

In this section, we are interested in studying the long-range interaction encoding in the attention matrices. We are going to pass through each model about 200 test graphs and compute metrics over the attentional matrices found at each layer. We started with λ_1 since it represents such a central value to this work, and we continue with a study of the

diameters, and commute times over the diameters of the attention coefficient matrices.

5.3.1 λ_1 distribution

In the following experiments, we computed the λ_1 coefficient over As we can see in Tables 5.2 and 5.3, Peptides start with 10 times smaller λ_1 values than the ZINC, but end up with attention that has much bigger λ_1 values in the first and last layers (almost 1 the graphs are almost fully connected with equal edge weights). Given that the first and last layers are going through a lot of informational load (i.e. many edges give many messages), it is not surprising that the middle layers were weakly connected.

From Figure 5.5 in the ZINC case, it looks like they seek structure, also from Table 5.1 are stronger correlated with the adjacency matrix, which could mean that these layers were supposed to surprise structure.

Table 5.2: λ_1 Values Peptides: λ_1 at layers 1 and 5 has values very close to 1, which means the graphs are almost fully connected and the weights or the edges are almost fully equal.

λ_1 values over Peptides						
Model	Init. A	Attention Layer				
		1	2	3	4	5
GT	0.0025	0.9912	0.8285	0.8409	0.9074	0.9999
GT + conv(A)	0.0025	0.9981	0.9077	0.7775	0.8381	0.9995

Table 5.3: λ_1 Values ZINC: Big values of λ_1 over the layers 1, 4, and 5, with the fifth layer almost fully connected with equal weights. For $+conv(A)$, we can see a general increase in λ_1 over the layers, meaning that as we feed into the initial adjacency into the pipeline, the attention focuses even more on long range interactions.

$\lambda_1$values over ZINC						
Model	Init. A	Attention Layer				
		1	2	3	4	5
GT	0.0240	0.9674	0.9216	0.8758	0.9816	0.9958
GT + conv(A)	0.0240	0.9791	0.9511	0.9202	0.9814	0.9932

5.3.2 Diameter distribution

We proceed with investigating the diameter of attention coefficient matrices within the context of our study. In the case of unweighted graphs, the way it's computed is the following. Taking all shortest paths between any pair of nodes the algorithm returns the

longest shortest path it had met. For weighted graphs, the algorithm mainly does the same thing but the weights on the edges represent distances, and the shortest path between two nodes will be the path that has the smallest distance, rather than the smallest number of nodes encountered. Nevertheless, it is important to note that the weights assigned to the edges represent distances rather than traditional weights. In this case, smaller distance values indicate closer proximity between nodes, while larger weight values imply that the nodes are sharing more information with each other. As a result, we found it necessary to apply a transformation to the attention coefficient matrices. The transformation we picked is $1/(A_{att} + 0.0001)$. We added $+0.0001$ because we wanted to transform edges with value 0, in edges that are extremely long, more precisely that have a distance of 10000. Sometimes the attention matrices would be disconnected, a situation which would cause the diameter to be infinity.

Table 5.4: Diameter Values Peptides: We can see the same progression as we saw in λ_1 case however it is hard to believe the diameter over the first layer, is larger than the initial adjacency, this happens because we haven't scaled properly our values.

Diameter values over Peptides						
Model	Init. A	Attention Layer				
		1	2	3	4	5
GT	60.2631 ± 31.79	160.22 ± 75.23	302.57 ± 157.77	348.77 ± 168.78	258.69 ± 148.82	163.52 ± 84.11
GT + conv(A)	60.26 ± 31.79	331.78 ± 170.86	321.11 ± 166.25	247.99 ± 192.12	390.64 ± 270.69	166.47 ± 85.96

Table 5.5: Diameter Values ZINC: We can see the same progression as we saw in the λ_1 case however, just like in the Peptides case we think that the values from the attention haven't been scaled properly.

Diameter values over ZINC						
Model	Init. A	Attention Layer				
		1	2	3	4	5
GT	12.24 ± 2.53	49.64 ± 42.82	45.69 ± 10.41	66.98 ± 23.18	118.98 ± 164.78	71.01 ± 70.03
GT + conv(A)	12.24 ± 2.53	42.19 ± 22.68	41.81 ± 9.77	41.80 ± 8.27	43.83 ± 14.18	48.39 ± 28.49

From Tables 5.4, and 5.5 we can see that diameter doesn't seem to be a very stable metric.

We can notice that for Peptides, in Table 5.4, the diameter values are smaller in general, which means the graph formed by the attention mechanism seems to be more connected by a stronger edge. This choice of connectivity is reflected in 5.2 as well. We think the problem is that values are fairly small in the attention, so we want to scale them so that we get some edges with values 1, or smaller. We are going to try another way to process the diameter. And that is $1/(A_{att}/max(A_{att}) + 0.0001)$. This way the Diameter of the initial adjacency matrix A will remain the same because $1/(A/max(A)+0.0001) = 1/(A+0.0001)$ because all values in A are 1, so $max(A) = 1$ as well.

Table 5.6: Scaled Diameter Values Peptides: For Peptides, it is worth noticing that besides obtaining almost fully connected with equal weights at the last layer, is that at layer 3 and 4 values can get bigger than the diameter of the initial adjacency

Diameter values over Peptides						
Model	Init. A	Attention Layer				
		1	2	3	4	5
GT	60.26 ± 31.79	2.76 ± 0.34	63.67 ± 32.28	99.26 ± 58.65	3.57 ± 1.70	1.00 ± 0.00
GT + conv(A)	60.26 ± 31.79	4.88 ± 0.96	30.59 ± 24.65	70.97 ± 52.14	102.21 ± 69.40	1.04 ± 0.01

Table 5.7: Scaled Diameter Values ZINC: Something really remarkable about diameter values in the ZINC case is that all of them fall under the diameter of the initial graph.

Diameter values over ZINC						
Model	Init. A	Attention Layer				
		1	2	3	4	5
GT	12.25 ± 2.53	4.18 ± 2.09	10.19 ± 5.92	8.31 ± 3.77	4.27 ± 1.99	5.92 ± 3.26
GT + conv(A)	12.25 ± 2.53	11.62 ± 18.80	11.58 ± 4.75	15.39 ± 8.23	23.53 ± 28.56	7.18 ± 9.64

We think a really important takeaway from these metrics is the diameter over the attention at layer 5, in the case of Peptides, where it is around 1 with a very small error. That means the graphs are fully connected with almost equal edges in Table 5.6. This aligns with the λ_1 value coming from table 5.2.

For ZINC, we can see that the diameters of the attention coefficient for $GT + conv(A)$

are bigger than for GT.

5.3.3 Commute Times Distribution

Computing commute times on weighted graphs have been defined before in [51], we will summarize Proposition 1. In [51], the main focus of the authors defines commuting times in the class of *geometric graphs*, where they consider a set of points $X_1, X_2, ...X_n \in \mathbb{R}^d$, where the points will form the vertices in the graph. After that, they build edge structures on these nodes by using edges returned by the k-nearest neighbour algorithm. After that, they define a similarity measure $k(X_i, X_j)$ with $k : \mathbb{R}^d : R^d \to \mathbb{R}_{\geq 0}$, which helps them define the weights of the edges with entries $w_{ij} = k(X_i, X_j)$. In the paper, they use Gaussian similarity with $k(a, b) = \exp(-||X_i - X_j||^2/\sigma^2)$. Given this weighted adjacency matrix, they compute the symmetric graph Laplacian, $L_{sym} = I - D^{-1/2}AD^{-1/2}$, and after the Roger-Penrose inverse $L^{\dagger}_{sym}$. e_i is a vector of size n which has i-th entry 1, and the rest 0, and d_i is just a scalar equal to the sum of weights on that row. The commute times C_{ij} can be computed in the following way:

$$C_{ij} = \text{vol}(G) \left\langle \frac{1}{\sqrt{d_i}} e_i - \frac{1}{\sqrt{d_j}} e_j, L^{\dagger}_{sym} \left(\frac{1}{\sqrt{d_j}} e_j - \frac{1}{\sqrt{d_i}} e_i \right) \right\rangle \tag{5.1}$$

Remark 5. *This formula actually has a typo, which follows the fact that one needs the same ordering left and right to the inner product to get the squares to have positive signs. We figured the signs were flipped in Equation 5.1, for the first term of the inner product. We marked in red where the mistake in their formula lies, and in blue our changes. Instead, it should be:*

$$C_{ij} = \text{vol}(G) \left\langle \frac{1}{\sqrt{d_j}} e_j - \frac{1}{\sqrt{d_i}} e_i, L^{\dagger}_{sym} \left(\frac{1}{\sqrt{d_j}} e_j - \frac{1}{\sqrt{d_i}} e_i \right) \right\rangle \tag{5.2}$$

Remark 6. *Opposite to the diameter case in which we had to apply a decreasing function to the edge weights, in this case as in the case of λ_1, no processing on the adjacency matrix was needed, as edges weights represent similarity in this case. Also as the diameter is computed on the graph Laplacian, which is $I - N$, symmetrically normalized matrices we also didn't need any scaling.*

It was a surprise to learn that the commute times did not follow the same curve diameters in Tables 5.9, 5.8 and λ_1 in Tables 5.2, 5.3. For λ_1 analysis if we remember, the first and the last layer had big values, which means the graphs generated by the attention were strongly connected, and for diameters if we remember the case of peptides in Table 5.6 where the diameter at the fifth layer was almost 1. We think that even the very weak signals in Figure 5.4 or 5.5, got normalized. The only interpretation of these tables could be comparative between the two tasks. The ratio between the commute times at attention

layers and at the initial adjacency in the case of ZINC is around 1/5, while for peptides is 1/25, which means low commute times were implemented more obviously in the Peptides case.

Table 5.8: Commute Times Values ZINC: While the values are quite uniform we can notice some variation over the standard deviation.

Commute Times values over ZINC						
Model	Init. A	Attention Layer				
		1	2	3	4	5
GT	204.26 ± 68.40	37.86 ± 41.21	42.65 ± 8.28	42.70 ± 9.68	43.03 ± 14.62	43.71 ± 9.91
GT + conv(A)	204.26 ± 68.40	42.40 ± 21.05	42.23 ± 8.20	43.19 ± 9.14	44.50 ± 21.36	43.00 ± 28.49

Table 5.9: Commute Times Values Peptides: Very uniform commute time values, even in the standard deviation.

Commute Times over Peptides						
Model	Init. A	Attention Layer				
		1	2	3	4	5
GT	7627 ± 7153.3	317.17 ± 180.76	315.99 ± 180.87	313.43 ± 181.07	317.11 ± 181.21	316.83 ± 181.06
GT + Conv(A)	7627 ± 7153.3	316.94 ± 181.09	316.80 ± 181.07	317.92 ± 183.93	323.27 ± 210.76	316.87 ± 181.10

In the upcoming section, we will present our conclusions based on the findings and analysis conducted in this study.

5.4 Attentional Approach Conclusion

In this study, we have explored the attentional approach of Graph Transformers in graph learning. Our analysis focused on understanding the behaviour and characteristics of attention coefficient matrices and their implications for graph learning pipelines. By addressing key questions related to the attention mechanism in transformers, we have gained valuable insights into its functioning and its relationship with the graph's topology.

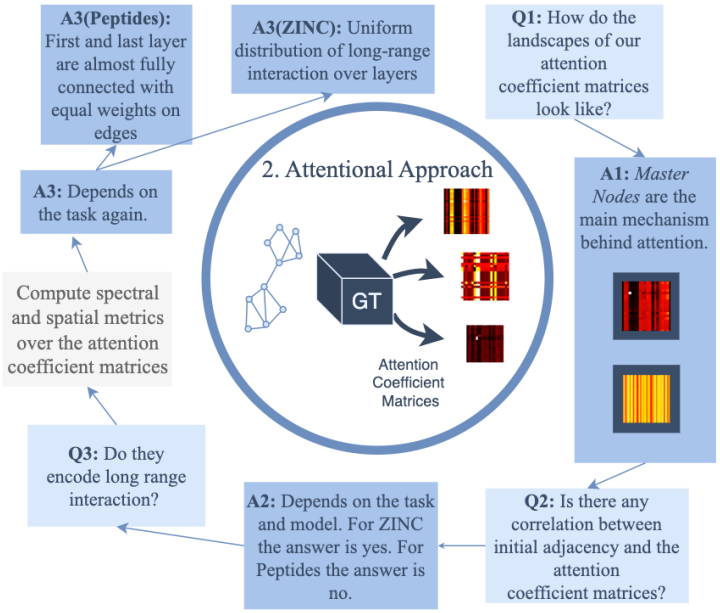

Figure 5.6: Attentional Approach Conclusion Schema

Through our investigation, we have made several noteworthy findings. Firstly, we discovered that the attention mechanism in Graph Transformers heavily relies on "master nodes," which act as central hubs for information exchange within the graph. These master nodes play a crucial role in capturing and propagating important information throughout the graph.

Secondly, we examined the correlation between attention coefficient matrices and the initial adjacency matrix. Our results showed that the level of correlation varied depending on the specific task. In the case of the ZINC dataset, which exhibits a structure-based nature, we observed a significant correlation between attention and the initial adjacency matrix. However, for the Peptides dataset, there was a lack of substantial correlation. This suggests that attention coefficients in transformers do not solely focus on capturing the graph's structural characteristics but also capture stable forms of long-range interactions.

Lastly, we employed metrics for measuring long-range interactions, such as diameters, spectral gap, and commute times, to analyze the attentional rewirings. Our analysis revealed that in the Peptides dataset, the attention extracted from the first and last layers formed nearly fully connected graphs, with equal weights on all edges. In contrast, the attention coefficient layers for the ZINC dataset exhibited a more uniform distribution of the long-interaction metrics over the layers.

These findings provide valuable insights into the behaviour of attentional approaches in graph transformers. They highlight the importance of understanding the role of attention in capturing and propagating information within the graph, as well as the influence of graph topology on attentional rewirings.

Chapter 6

Conclusion

In this study, we have explored two approaches, namely Variational Graph Autoencoder (VGAE) and Graph Transformer, to understand and address the challenges of over-squashing and rewiring.

The VGAE-based approach focused on alleviating over-squashing by incorporating the spectral gap (λ_1) of the generated graph into the loss function. Our experiments demonstrated that VGAEs can effectively learn to generate rewired adjacency matrices that mitigate bottlenecks within the graph. The rewirings yielded improved graph connectivity, enabling better information flow and enhancing performance on supervised tasks. Notably, in the case of the Peptides-struct dataset, which emphasizes long-range interactions, our models performed nearly as well as Transformer-based architectures. Additionally, we observed that generating extremely bottlenecked graph rewirings did not significantly impact performance on both Peptides-struct and ZINC datasets. Which requires further research.

Furthermore, we delved into the attentional mechanisms of Graph Transformers, which learn soft adjacency matrices through message passing. By analyzing the attention coefficient matrices and computing topological graph metrics, we uncovered intriguing findings. The attention mechanism in transformers predominantly relies on *master nodes* that are connected to every other node, serving as central hubs for information exchange. Regarding the correlation between attention and the initial adjacency matrix, we observed task-dependent variations. While there was some correlation in the case of the ZINC dataset, aligning with its structure-based nature, the attention coefficient matrices for the Peptides-struct dataset exhibited minimal correlation, indicating that they primarily encode stable forms of long-range interactions. Our analysis of the coefficient adjacency matrices revealed that the attention extracted from the first and last layers of Peptides-struct forms nearly fully connected graphs with equal edge weights, while the attention coefficient layers for ZINC exhibit a more balanced structure.

In conclusion, our studies revealed that the attention coefficient matrices learned by the

attentional approach exhibited characteristics of algorithmic adjacency matrices, where nodes were selected as *master nodes*. Conversely, in the Variational study, as the value of α increased and less bottlenecked graphs were generated, we observed an increase in loss, suggesting that the models struggled to comprehend the actual information provided to them. This could suggest that algorithmic rewirings might be the solution moving forward. However, in the next section, we are going to introduce another differentiable variational technique that might alleviate this problem of the model struggling to understand the rewired graphs generated by the VGAE.

6.1 Further work

We propose two avenues for future work based on the findings and limitations of our study.

Firstly, in the realm of differentiable rewiring methods, we suggest incorporating the Variational Graph Autoencoder (VGAE) into the supervised learning pipeline. By training the supervised model in conjunction with the VGAE, the proposed rewiring method would be "differentiable all the way," allowing the generated rewirings to adapt in a way that the supervised model can effectively interpret. This integration has the potential to enhance the performance of the supervised model by leveraging the benefits of rewiring while maintaining interpretability and coherence.

Secondly, regarding the attention approach employed in the Graph Transformer architecture, we have developed an explainability tool to understand the expressivity of the attention mechanism. This tool can be extended beyond our specific architecture and applied to other models and tasks, facilitating a deeper analysis of attentional coefficients in the layers of the graph transformer. To broaden its applicability, we intend to enhance the tool to support multi-head attention and accommodate different architectures. This extension will provide the research community with a valuable resource to scrutinize and gain insights into the attention mechanisms employed in graph-related tasks.

Bibliography

[1] Ralph Abboud, Radoslav Dimitrov, and İsmail İlkan Ceylan. "Shortest path networks for graph property prediction". In: *arXiv preprint arXiv:2206.01003* (2022).

[2] Uri Alon and Eran Yahav. "On the bottleneck of graph neural networks and its practical implications". In: *arXiv preprint arXiv:2006.05205* (2020).

[3] Adrian Arnaiz-Rodriguez et al. "DiffWire: Inductive Graph Rewiring via the Lovasz Bound". In: *arXiv preprint arXiv:2206.07369* (2022).

[4] Pradeep Kr Banerjee et al. "Oversquashing in GNNs through the lens of information contraction and graph expansion". In: *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE. 2022, pp. 1–8.

[5] Pablo Barceló et al. "The logical expressiveness of graph neural networks". In: *8th International Conference on Learning Representations (ICLR 2020)*. 2020.

[6] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[7] Beatrice Bevilacqua et al. "Equivariant subgraph aggregation networks". In: *arXiv preprint arXiv:2110.02910* (2021).

[8] Cristian Bodnar et al. "Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 18527–18541.

[9] Cristian Bodnar et al. "Weisfeiler and lehman go cellular: Cw networks". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 2625–2640.

[10] Michael M Bronstein et al. "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges". In: *arXiv preprint arXiv:2104.13478* (2021).

[11] Rickard Brüel-Gabrielsson, Mikhail Yurochkin, and Justin Solomon. "Rewiring with positional encodings for graph neural networks". In: *arXiv preprint arXiv:2201.12674* (2022).

[12] Chen Cai and Yusu Wang. "A note on over-smoothing for graph neural networks". In: *arXiv preprint arXiv:2006.13318* (2020).

[13] Fan RK Chung. *Spectral graph theory*. Vol. 92. American Mathematical Soc., 1997.

[14] Gabriele Corso et al. "Principal neighbourhood aggregation for graph nets". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13260–13271.

[15] Antonia Creswell et al. "Generative adversarial networks: An overview". In: *IEEE signal processing magazine* 35.1 (2018), pp. 53–65.

[16] Andreea Deac, Marc Lackenby, and Petar Veličković. "Expander graph propagation". In: *arXiv preprint arXiv:2210.02997* (2022).

[17] Austin Derrow-Pinion et al. "Eta prediction with graph neural networks in google maps". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 3767–3776.

[18] Francesco Di Giovanni et al. "On Over-Squashing in Message Passing Neural Networks: The Impact of Width, Depth, and Topology". In: *arXiv preprint arXiv:2302.02941* (2023).

[19] Vijay Prakash Dwivedi and Xavier Bresson. "A generalization of transformer networks to graphs". In: *arXiv preprint arXiv:2012.09699* (2020).

[20] Vijay Prakash Dwivedi et al. "Benchmarking graph neural networks". In: (2020).

[21] Vijay Prakash Dwivedi et al. "Graph neural networks with learnable structural and positional representations". In: *arXiv preprint arXiv:2110.07875* (2021).

[22] Vijay Prakash Dwivedi et al. "Long range graph benchmark". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 22326–22340.

[23] Fabrizio Frasca et al. "Sign: Scalable inception graph neural networks". In: *arXiv preprint arXiv:2004.11198* (2020).

[24] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. "Diffusion improves graph learning". In: *Advances in neural information processing systems* 32 (2019).

[25] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[26] Lorenzo Giusti et al. "Graph Convolutional Networks With Autoencoder-Based Compression And Multi-Layer Graph Learning". In: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2022, pp. 3593–3597.

[27] Benjamin Gutteridge et al. "DRew: Dynamically Rewired Message Passing with Delay". In: *arXiv preprint arXiv:2305.08018* (2023).

[28] Junheng Hao et al. "P-companion: A principled framework for diversified complementary product recommendation". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 2517–2524.

[29] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

[30] Weihua Hu et al. "Strategies for pre-training graph neural networks". In: *arXiv preprint arXiv:1905.12265* (2019).

[31] Jaehyeong Jo, Dongki Kim, and Sung Ju Hwang. "Graph Generation with Destination-Driven Diffusion Mixture". In: *arXiv preprint arXiv:2302.03596* (2023).

[32] Chaitanya Joshi. "Transformers are Graph Neural Networks". In: *The Gradient* (2020).

[33] Kedar Karhadkar, Pradeep Kr Banerjee, and Guido Montúfar. "FoSR: First-order spectral rewiring for addressing oversquashing in GNNs". In: *arXiv preprint arXiv:2210.11790* (2022).

[34] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[35] Thomas N Kipf and Max Welling. "Variational graph auto-encoders". In: (2016).

[36] Devin Kreuzer et al. "Rethinking graph transformers with spectral attention". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 21618–21629.

[37] Qimai Li, Zhichao Han, and Xiao-Ming Wu. "Deeper insights into graph convolutional networks for semi-supervised learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[38] Qi Liu et al. "Constrained graph variational autoencoders for molecule design". In: *Advances in neural information processing systems* 31 (2018).

[39] Laszlo Lovasz. "Random walks on graphs". In: *Combinatorics, Paul erdos is eighty* 2.1-46 (1993), p. 4.

[40] Gregoire Mialon et al. "Graphit: Encoding graph structure in transformers". In: *arXiv preprint arXiv:2106.05667* (2021).

[41] Christopher Morris et al. "Weisfeiler and leman go neural: Higher-order graph neural networks". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 4602–4609.

[42] Hoang Nt and Takanori Maehara. "Revisiting graph neural networks: All we have is low-pass filters". In: *arXiv preprint arXiv:1905.09550* (2019).

[43] Ladislav Rampášek et al. "Recipe for a general, powerful, scalable graph transformer". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 14501–14515.

[44] Martin Simonovsky and Nikos Komodakis. "GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders". In: *Artificial Neural Networks and Machine Learning – ICANN 2018*. Ed. by Věra Kůrková et al. Cham: Springer International Publishing, 2018, pp. 412–422. ISBN: 978-3-030-01418-6.

[45] Yang Song et al. "Score-based generative modeling through stochastic differential equations". In: *arXiv preprint arXiv:2011.13456* (2020).

[46] Jonathan M Stokes et al. "A deep learning approach to antibiotic discovery". In: *Cell* 180.4 (2020), pp. 688–702.

[47] Jake Topping et al. "Understanding over-squashing and bottlenecks on graphs via curvature". In: *arXiv preprint arXiv:2111.14522* (2021).

[48] Francisco Vargas, Teodora Reu, and Anna Kerekes. "Expressiveness Remarks for Denoising Diffusion Models and Samplers". In: *arXiv preprint arXiv:2305.09605* (2023).

[49] Petar Veličković. "Message passing all the way up". In: *arXiv preprint arXiv:2202.11097* (2022).

[50] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[51] Ulrike Von Luxburg, Agnes Radl, and Matthias Hein. “Hitting and commute times in large random neighbourhood graphs”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1751–1798.

[52] Zonghan Wu et al. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[53] Keyulu Xu et al. “How powerful are graph neural networks?” In: (2018).

[54] Chengxuan Ying et al. “Do transformers really perform badly for graph representation?” In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28877–28888.

[55] Rex Ying et al. “Graph convolutional neural networks for web-scale recommender systems”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 974–983.

[56] Jie Zhou et al. “Graph neural networks: A review of methods and applications”. In: *AI open* 1 (2020), pp. 57–81.

Appendix A

Technical details, proofs, etc.

A.1 Computational Resources and Code Assets

All model experiments were run on V100 and T4 GPUs from Google Colab. The code base was from scratch and built upon PyTorch Geometric.

The datasets ZINC was provided by PyTorch Geometric, and Peptides by codebase available from [22].

A.2 Types of MPNNs

In this section, we give a taste of the various MPNNs that the field has created, to give the reader some intuition of what forms the functions ψ, ϕ, and $\oplus$ might take. We will first introduce probably the most popular type of GNN the Graph Convolutional Network (GCN) [34], and then Graph Isomorphism Network (GIN) [53] and finish up with Principal Neighbourhood Aggregation (PNA) [14]. There are many other types of GNNs, like Graph Attention Networks (GAT) from [50], or Cellular Isomorphism Networks [9] but we won't introduce this further as they are not used in this work.

Graph Convolutional Networks Graph Convolutional Networks (GCNs) [34] use a convolutional layer that leverages a message-passing scheme to process graph-structured data. In this scheme, the feature information of each node is updated by aggregating the features of its neighbouring nodes scaled by the weight of the edge that joins them. Formally:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} c_{uv}\mathbf{x}_v\right) \tag{A.1}$$

Here c_{uv} is equal to entry uv of the normalized adjacency matrix. This operation can be viewed as a first-order approximation of spectral graph convolutions.

Graph Isomorphism Networks Graph Isomorphism Networks (GINs) [53] was proposed to tackle the problem of distinguishing non-isomorphic graphs, which is a shortcoming in some graph neural networks (GNNs). They are designed to capture a form of the Weisfeiler-Lehman (WL) graph isomorphism test, which is a heuristic for graph isomorphism. The WL test works by iteratively updating the label of a node with a hashed combination of its own label and the sorted list of its neighbours' labels, and this process continues until no new labels can be assigned. If at any iteration, the label sequences of two graphs match, they are said to be potentially isomorphic.

The GINs implement a learnable and local version of the WL test. The key idea behind GINs is to incorporate the information of a node's neighbourhood into the node itself. They propose a simple modification to the standard GNN layer that ensures injectivity (i.e., different graphs will map to different embeddings), making it theoretically as powerful as the WL test. In the GINs update rule, each node's new feature vector is computed as a nonlinear function (an MLP) of the node's current feature vector and the sum of its neighbours' feature vectors. We can translate this rule to fit into the general message-passing neural network (MPNN) framework as in Eq. 3.1, with the ϕ function representing the update function and the ψ function representing the message function.

In the context of GINs, the rule that updates nodes' representation can be defined as follows:

$$\mathbf{h}_v = MLP\left((1+\epsilon)\cdot \mathbf{x}_u + \sum_{v\in\mathcal{N}_u} \mathbf{x}_v\right) \tag{A.2}$$

Principal Neighbourhood Aggregation Principal Neighbourhood Aggregation (PNA) for graph nets [14] proposes a message-passing neural network where multiple aggregators with multiple degree scalers are used. In the paper, they prove their main result by saying that in order to discriminate between multisets of size n, the values coming from a node neighbourhood which belong to $\mathbb{R}$, at least n aggregators are needed.

$$\mathbf{x}_i' = \phi\left(\mathbf{x}_i, \bigoplus_{j\in\mathcal{N}(i)} \psi\left(\mathbf{x}_i, \mathbf{x}_j\right)\right), \tag{A.3}$$

$$\bigoplus = \underbrace{\begin{bmatrix} 1 \\ S(\mathbf{D}, \alpha = 1) \\ S(\mathbf{D}, \alpha = -1) \end{bmatrix}}_{\text{scalers}} \otimes \underbrace{\begin{bmatrix} \mu \\ \sigma \\ \max \\ \min \end{bmatrix}}_{\text{aggregators}} \tag{A.4}$$

As we can see in Equation A.4 the aggregator $\oplus$ is not 1-dimensional anymore, it will

compute the mean, standard deviation, max, and min, at different scales, obtaining a map that will return a vector of size 12.