TU WIEN Informatics

Novel Graph Representations for Improved Transformer-Based Graph Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Data Science

eingereicht von

Terezia Olsiakova
Matrikelnummer 12331438

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dipl.-Ing. Dr.techn. Matthias Lanzinger, BSc
Mitwirkung: Assistant Prof. Dr.techn. Stefan Neumann, BSc MSc

Wien, 3. September 2025

Terezia Olsiakova　　Matthias Lanzinger

Novel Graph Representations for Improved Transformer-Based Graph Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Data Science

by

Terezia Olsiakova
Registration Number 12331438

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dipl.-Ing. Dr.techn. Matthias Lanzinger, BSc
Assistance: Assistant Prof. Dr.techn. Stefan Neumann, BSc MSc

Vienna, September 3, 2025

Terezia Olsiakova | Matthias Lanzinger

Erklärung zur Verfassung der Arbeit

Terezia Olsiakova

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 3. September 2025

Terezia Olsiakova

Acknowledgements

My biggest thanks go to my supervisors, Matthias and Stefan. Thank you for your guidance and support throughout the whole process of working on this thesis. Your enthusiasm and passion for research were truly inspiring and motivated me to dig deeper and be more inquisitive in the research for my thesis. I am very grateful for your feedback and everything I have learned along the way.

Thank you to my family and friends for supporting me during my studies and for always being there for me.

Kurzfassung

Graphenlernen ist ein Teilgebiet des machinellen Lernens, das sich auf das Lernen und Extrahieren von Informationen aus Graphdaten konzentriert. In unserer Arbeit konzentrieren wir uns auf Graphen von Molekülen und untersuchen, wie die Struktur dieser Graphen kodiert werden kann. Die Kodierungen unterstützen dann verschiedene Aufgaben in Machine Learning, wie beispielsweise die Klassifizierung von Molekülfunktionen oder die Vorhersage ihrer Wasserlöslichkeit.

In den letzten Jahren hat die Transformer-Architektur im Bereich der Verarbeitung natürlicher Sprache große Erfolge erzielt und wurde auch für das Graphenlernen angepasst, wo sie auch eine sehr hohe Leistung erreicht hat. Graphen erfassen Informationen jedoch ganz anders als Textdaten. Das bringt die Herausforderung, wie der Graph kodiert werden muss, damit die Transformer-Architektur ihn lernen kann. Ein besonders wichtiges Konzept des Graphenlernens mit Transformer-basierten Architekturen ist die Positionskodierung von Graphen. Die Positionskodierung eines Graphen ist eine Matrixdarstellung der Graphenstruktur, die in den Transformer integriert wird. Es hat sich gezeigt, dass die Qualität dieser Kodierung einen direkten Einfluss auf die Leistung der Modelle hat. Daher eröffnet die Untersuchung und Verbesserung der Graphenkodierungen die Möglichkeit, das Lernen auf Graphen mit Transformer-Architekturen insgesamt zu verbessern.

In unserer Arbeit untersuchen wir zwei Methoden zur Kodierung von Graphstrukturen und integrieren diese Graphkodierungen in den Transformer, um das Lernen zu verbessern. Im Vergleich zu den derzeit verwendeten Methoden untersucht diese Arbeit einen neuen Weg zur Kodierung von Graphstrukturen auf der Grundlage der Faktorisierung der Graph-Adjazenzmatrixen. Die neuartige Kodierung bietet eine Verbesserung gegenüber dem Lernen ohne Positionskodierung und erreicht vergleichbare Ergebnisse zu den besten Methoden auf einem von den drei untersuchten Datensätzen. Damit zeigt sie einen vielversprechenden neuen Weg für die Berechnung von Graphkodierungen für das Graphenlernen mit Transformer-basierten Architekturen auf.

Abstract

Graph learning is a subfield of machine learning that focuses on learning and extracting information from graph data. In our work, we focus on graphs that represent molecules and study how the structure of these graphs can be encoded to aid in machine learning tasks such as classifying molecules based on their function or predicting their water solubility coefficients.

In recent years, the Transformer architecture has seen great success in the natural language processing domain and has also been adapted for graph learning, reaching state-of-the-art performance. However, graphs capture information very differently compared to textual data, which poses the challenge of how to encode the graph so that the transformer architecture can learn it. One particular key concept of graph learning with transformer-based architectures is the graph positional encoding. The positional encoding of a graph is a matrix representation of the graph structure, which is then a part of the input for training the transformers. It is shown that the quality of this encoding directly impacts the performance of the models, and hence studying and improving the graph encodings opens the possibility for improving the learning overall.

In our work, we study two methods for encoding graph structure and integrate these graph encodings into the transformers, aiming to improve learning. Compared to the currently used methods, this work explores a new path of encoding graph structure based on factorizing the graph adjacency matrix. The novel encoding provides improvement over baselines and matches the state-of-the-art method on one of the studied datasets, showing a promising new path for the computation of graph encodings for graph learning with transformer-based architectures.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

Transformers [VSP+17] have recently demonstrated great performance not only in natural language processing tasks but also in other areas like recommender systems or even graph learning. In the field of graph learning, we perform standard machine learning tasks such as regression or classification, given that the input to these tasks is a collection of nodes and edges.

A common framework that is used to perform such tasks is Graph Neural Networks (GNNs) [GSR+17]. However, particularly for graphs of molecules, Transformers are shown to outperform the Graph Neural Networks [DRG+22], [RGD+22], [DB20], since in contrast to the message passing mechanism used in GNNs, the attention mechanism of Transformers allows the nodes to attend to each other globally throughout the whole graph instead of just passing information locally [RGD+22].

Transformers are inherently unaware of any structure in the data [DLL+22]. Therefore, just like in text processing, where the position of each word in the sentence is passed to the encoder/decoder in the form of positional encoding, the same is done when learning the graph structure using Graph Transformers. Moreover, the structure of the graph holds a lot of information and hence makes the positional encoding even more challenging as the expressivity of the encoding impacts the generalizability of the models [RGD+22]. Thus, the need arises to provide the Transformer with such an encoding of the graph that is best able to capture the structure and aid in the learning process to improve the performance of the downstream machine-learning task.

1.2 Aim of the Work

This thesis aims to explore a new approach to the structural encoding of graphs that can be integrated into the Graph Transformer to improve the performance of machine-learning tasks, with a focus on molecular datasets. The main goal is to address the lack of methods currently available in the positional encoding module of the Graph Transformer.

The biggest contrast between existing methods and the proposed one is the focus on reconstructing the adjacency matrix of the input graph network from the computed encoding. This approach stems from the belief that the adjacency matrix captures the structure of the graph and hence is directly tied to the structural encoding. The methods proposed are based on the factorization of the binary graph adjacency matrix and allow for its reconstruction from the resulting encoding. This approach hopes to bridge the gap between the encoding method's interpretability and theoretical soundness and the performance of the graph learning done by the Transformer.

Since the proposed approach of creating the node encodings builds on applying the mathematical observations made on binary matrices to the adjacency matrices of the graphs, it brings a theoretically well-motivated method.

1.3 Overview and Methodological Approach

Throughout the experiments, three benchmark datasets will be used to study the effect of the different encoding methods as well as to perform the machine-learning tasks. Table 1.1 provides an overview of the relevant dataset characteristics of the chosen datasets.

dataset	number of graphs	nodes per graph	task	metric
ZINC	12.000	9-37	regression	mean absolute error (MAE)
Peptides-func	15.535	8-444	classification	average precision (AP)
CIFAR10	60.000	85-150	classification	accuracy (ACC)

Table 1.1: Characteristics of the datasets used in the experiments.

The chosen datasets are the following:

- ZINC [SI15, GDH+16] - dataset with molecular compounds and their solubility; we use the benchmarking subset of the dataset [DJL+20];
- Peptides-func [DRG+22] - a molecular dataset of peptides which displays long-range interactions between the nodes in the graphs, and each peptide is labeled based on its function;
- CIFAR10 [DJL+20] - dataset from the image classification domain, which offers different graph structures compared to molecular data, and each graph is constructed by aggregating and labeling areas of the image into hyperpixels.

The work is split into two main parts. In the first part, two methods for factorizing the adjacency matrices of the graphs are studied with the aim of understanding how the methods capture the structure of the graphs. The outcomes of the analysis from this part are then applied to compute the graph encodings for each dataset. The encodings are then integrated as part of the input for the Transformer graph learning. The second part is then the training of the Transformer itself to perform the machine-learning task at hand. In the second part, the performance of the model with different types of positional encodings is evaluated. The following two sections outline the methodology of each part in more detail.

1.3.1 Factorizing Graph Adjacency Matrices

Two methods for factorizing graph adjacency matrices are studied. The first method is the truncated singular value decomposition (tSVD) combined with applying the rounding at the matrices' rounding rank [NGM16]. In this method, for a rank k, the input matrix is factorized into two matrices L and R; then the original adjacency matrix is reconstructed after rounding is applied. The second method is the Logistic Principal Component Analysis (LPCA), which solves an optimization problem by minimizing the Frobenius error between the adjacency matrix and its low-dimensional representation of rank k.

Afterwards, an analysis of the results of tSVD and LPCA is carried out to gain insights into how the methods differ in terms of relative reconstruction error, runtime, and the size of the k required to achieve perfect reconstruction. The choice of an appropriate k per dataset is also studied based on the quality of the produced results, namely their relative reconstruction error, which shows that the factorization captures enough of the structure of the graph. On the other hand, since the k determines the dimension of the computed node embedding, other factors have to be considered, such as k being sufficiently small for effective integration in the Graph Transformer.

1.3.2 Transformer Graph Learning

Once the node encodings are computed, the Graph Transformer is trained in multiple experiment runs. The first experiment will be run without any positional encoding to obtain a baseline performance metric for each dataset. Then, additional runs with the new encodings integrated into the Graph Transformer will be done. In this step, multiple configurations will be tried, including tuning the hyperparameters of the graph transformer or recomputing the encoding with a different choice of parameter k and different seeds for initialization of the matrices, where relevant (LPCA), to obtain the best results.

Finally, the performance of the new encoding is evaluated based on the relevant performance metrics against the benchmarks of the current state-of-the-art positional encodings used in the Graph Transformers. The performed machine-learning tasks and evaluation metrics per dataset are summarized in Table 1.1.

1.4 Research Questions

To guide the research process, we define the following three research questions:

1. Which of the two methods, tSVD or LPCA, produces a lower relative reconstruction error for a dimension k per dataset?
2. What dimension k is needed to achieve perfect reconstruction of the adjacency matrices for all of the graphs in each dataset?
3. How does the new encoding method compare to using no encoding and other encodings (RWSE, LapPE, MoSE) in terms of:
   a) accuracy for classification on CIFAR10,
   b) average precision for classification on Peptides-func,
   c) mean absolute error for regression on ZINC?

1.5 Structure of the Thesis

Chapter 2 gives the necessary background knowledge, explains key concepts related to the experiments, and lays the ground for understanding the work in a broader context. The key concepts explained here are factorization methods for binary matrices, graph learning, and Graph Transformers.

In Chapter 3, the first part of the thesis is described. Here, the main topic is the factorization of binary adjacency matrices. The chapter details the experiment setup, implementation, and summarizes key findings, results, and challenges encountered along the way. It contains the answers to the first two research questions.

Following that, Chapter 4 takes the insights and outputs achieved in the previous chapter and focuses on integrating them into the Graph Transformer architecture. The experiment runs and the various approaches taken are described here, including the final results and answer to the third research question.

Finally, the key findings, contributions, and insights gained from this work are summarized in Chapter 5.

CHAPTER 2

Background and Key Concepts

This chapter gives an introduction to the key concepts used in this work. In the first part of the chapter, the basic concepts of graphs, their representation, and the factorization of binary matrices are laid out. Then, the Graph Neural Network and Graph Transformer architectures are introduced. Finally, the two parts are brought together in the form of positional graph encodings used in Transformer-based graph learning. In this last part, we also give an overview of state-of-the-art graph encodings.

2.1 Factorizing a Binary Matrix

The matrix factorization methods explored in this work take as input binary matrices. To understand how these relate to the graph representation, we first define the graph structure and particularly its representation in the form of an adjacency matrix.

A graph is formally defined as a pair $G = (V, E)$, where V is the set of nodes or vertices, and E is the set of all edges or links between nodes $E \subseteq V \times V$. Given the formal definition, there are various ways to represent the graph as a data structure. The option relevant for this work is representing the graph with the so-called *adjacency matrix* A. Then A has the size $|V| \times |V|$ and for each pair of nodes $u, v \in V$ the matrix entry is:

$$A_{uv} = \begin{cases} 1, & \text{if } (u, v) \in E. \\ 0, & \text{otherwise.} \end{cases}$$

Immediately, we observe that the adjacency matrix of a graph is inherently a binary square matrix. For completeness, if the edges in the graph were weighted, then the adjacency matrix no longer remains binary, and the range of its values is defined by the range of the weight function. In our work, we studied only unweighted graphs and hence adhere to the definition given above in the rest of this chapter.

Another observation that can be made is that the properties of the adjacency matrix reflect the properties of the graph. In our case, A is always symmetric since all graphs G are undirected graphs. Furthermore, A contains only zeros on the main diagonal because we assume that G contains no self-loops.

Finally, it is important to note that representing graphs as adjacency matrices does not provide a single unique valid representation. For example, given a list of edges between the nodes in the graph, which is how the graphs are defined in our dataset, we can construct multiple valid binary matrices that represent the same graph. The process to construct these matrices is as follows: given a list of nodes and edges, construct an adjacency matrix A, so that each row and column corresponds to a node's neighbourhood in the graph. However, by definition, V and E are unordered sets. Thus, there is no defined order of the rows or columns in the matrix. With that, it is possible to construct various binary matrices that all represent the structure of one graph.

2.1.1 Low-rank Approximation

Now that we have established the representation of the graph structure in the form of a binary adjacency matrix, we explore factorization methods that take as input a binary matrix and approximate it by a matrix of lower rank. Low-rank approximation is formally defined as: given the input matrix $A \in \{0,1\}^{n \times n}$, find such a matrix $\widehat{A}$ that is of lower rank than A and minimizes the Frobenius norm between the two matrices:

$$\|\widehat{A} - A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |\widehat{a}_{ij} - a_{ij}|^2}.$$

2.1.2 Matrix Rounding Rank

Before we introduce the two matrix factorization methods, the last key concept is the matrix rounding, which plays a role in the reconstruction of the original adjacency matrix once the encodings are computed. First, we define the rounding function [NGM16]:

$$\text{round}_\tau(x) = \begin{cases} 1, & \text{if } x \geq \tau, \\ 0, & \text{if } x < \tau. \end{cases}$$

where $\tau \in \mathbb{R}$ is the rounding threshold. This function is applied element-wise to a matrix to compute the rounded matrix. For a matrix A, its *rounding rank* is the smallest rank k for which any real-valued matrix rounds to A with respect to the given threshold τ.

In our case, our goal is to find matrices $L \in \mathbb{R}^{n \times k}$ and $R \in \mathbb{R}^{n \times k}$, such that after we apply rounding to the low-rank matrix $LR^\top$, we obtain the reconstructed adjacency matrix A. In other words we want that $\text{round}_\tau(LR^\top) = A$. The exact application of the rounding function with respect to the used method is explained in more detail in the following two sections, where each of the two decomposition methods is introduced.

2.1.3 Truncated Singular Value Decomposition

Singular value decomposition (SVD) is a method used in decomposing matrices that, for an arbitrary adjacency matrix $A \in \{0,1\}^{n \times n}$, results in the following decomposition:

$$A = U \Sigma V^\top,$$

where the $U \in \mathbb{R}^{n \times k}$ and $V \in \mathbb{R}^{n \times k}$ matrices are made up of the singular vectors of X and $\Sigma \in \mathbb{R}^{k \times k}$ contains the singular values on the main diagonal, organized from largest to smallest. It is possible to compute the SVD in polynomial time [HMT11].

We set $L = U$ and $R = \Sigma V^\top$, which gives us the two low-rank matrices $L, R \in \mathbb{R}^{n \times k}$. In addition, the method we are using is the truncated SVD, meaning only submatrices of the resulting L, R matrices are considered instead of the full decomposition, which allows for the desired dimensionality reduction. In truncated SVD, for a matrix of size $n \times n$ and a dimension k where $k < n$, we leave out (or set to zero) $n - k$ of the smallest singular values to obtain the lower-dimension decomposition of the input adjacency matrix [Han87].

Naturally, the dimension k can be thought of as a hyperparameter, as it is a subject of choice how much of a reduction we aim for. When combined with the rounding function, we can reconstruct the original adjacency matrix at some threshold τ used in the rounding. If the reconstructed matrix matches the adjacency matrix A, then k is an upper bound on the rounding rank of the matrix A.

On a final note, the SVD suffers from sign ambiguity, meaning the signs of the singular values can be flipped and still remain a valid SVD decomposition. In Section 2.4, we discuss what implications this has on the encoding once integrated into the Transformer, which has been illustrated on the Laplacian eigenvector encodings. These face the same issue. However, in that case, it is the ambiguous signs of the eigenvectors.

2.1.4 Logistic Principal Component Analysis

Principal Component Analysis (PCA) is another common technique for dimensionality reduction of data [CDS01, SSU03]. The generalized version of PCA suitable for binary input matrices is the logistic PCA (LPCA), which is based on maximizing the log-likelihood between the input matrix and the low-rank approximation [SSU03].

The goal is to minimize the error between the lower-dimensionality representation and the adjacency matrix, which can be achieved by iteratively minimizing the following loss function [CMST20]:

$$L = \sum_{i=1}^{n} \sum_{j=1}^{n} -\log \ell \left(\tilde{a}_{i,j} [LR^T]_{i,j} \right).$$

Here $\widetilde{A}$ is the shifted adjacency matrix $\widetilde{A} = 2A - 1$, where the 0's are replaced with -1's. After applying the rounding [NGM16], we want that $\text{round}_0(LR^\top) = A$. We round with the threshold $\tau = 0$, because all negative entries in the reconstructed matrix $LR^\top$ indicate that there is no edge between corresponding nodes, and non-negative entries indicate that there is an edge between them in the lower-dimensional representation.

2.2 Graph Neural Networks

Before introducing the Graph Transformer (GT) architecture, we briefly introduce Graph Neural Networks (GNNs) to contrast and compare the two frameworks and give a better intuition for the working of GTs. As we will see, the GNNs preceded the GTs, which then took the concepts already established in GNNs to build the new architecture on top of. Therefore, by first understanding the GNNs and their application in graph learning we build the foundations for the transformer architecture for graph learning, which is introduced later in the upcoming section.

Graph Neural Networks are a framework for learning graph representations. They are a diffusion model [SGT+09]. The intuition behind this framework is that the individual nodes in the graph carry some information and are able to share it with their neighbors based on the graph connectivity, then the neighbors of the neighbors do the same, and so on until it converges [SGT+09, HYL17], given that the node representations are initialized arbitrarily [LTBZ16]. This base framework has been adapted by many, introducing multiple variations of GNNs.

Gilmer et al. [GSR+17] define the Message Passing Neural Networks (MPNNs). Particularly in this work, the authors focused on the application in the quantum chemistry domain, predicting properties of organic molecules, aiming to make this architecture the standard in machine learning on molecules. They define the message-passing mechanism as follows [GSR+17]:

$$\begin{aligned} m_v^{(t+1)} &= \sum_{w \in N(v)} M_t(h_v^{(t)}, h_w^{(t)}, e_{vw}) \\ h_v^{(t+1)} &= U_t(h_v^{(t)}, m_v^{(t+1)}) \\ \widehat{y} &= R(\{h_v^{(T)} \mid v \in G\}) \end{aligned}$$

M_t is the message function and U_t is the update function. Then each message m_v for node $v \in V$ at time $t+1$ is computed as the sum of the message from time t from all neighbours $w \in N(v)$, where the message function is applied to the hidden node features of node v and the neighbour w, and the hidden state of the edge between them e_{vw}. The hidden state of the node is also updated in each iteration by applying the function to the state h_v from time t and the message $m_v^{(t+1)}$ computed in the current iteration. The readout function R is then used to compute the feature vector of the graph, where $h_v^{(T)}$ is the final hidden state of node v.

Li et al. [LTBZ16] modified the architecture proposed by Scarselli et al. [SGT+09] and introduced Gated Graph Neural Networks (GG-NNs) which employ the Gated Recurrent Unit [CvMG+14] in the update function. Using the message passing notation from Gilmer et al. [GSR+17], in this case, the message and update functions are the following:

$$M_t(h_v^{(t)}, h_w^{(t)}, e_{vw}) = A_{e_{vw}}\, h_w^{(t)}$$
$$U_t = \mathrm{GRU}(h_v^{(t)}, m_v^{(t+1)})$$

Here, A is a learned matrix representation of the edge labels, and GRU is the Gated Recurrent Unit.

Kipf and Welling [KW16] define the Graph Convolutional Networks (GCNs), which consist of multiple layers where the propagation rule is applied. The basis for this rule is in the spectral convolutions on graphs [HVG09]. This propagation rule can be reformulated into the message and update function as follows [GSR+17]:

$$M_t(h_v^{(t)}, h_w^{(t)}) = \tilde{A}_{vw}(\deg(v)\deg(w))^{-1/2} h_w^{(t)}$$
$$U_t(h_v^{(t)}, m_v^{(t+1)}) = \sigma\left((W^{(t)})^{\top} m_v^{(t+1)}\right)$$

where $\tilde{A} \in \mathbb{R}^{|V|\times|V|}$ is the adjacency matrix with an added self-loop, which is done by adding 1 along the main diagonal. The $W \in \mathbb{R}^{f\times f}$ matrix is the trainable weight matrix for the given layer, where f is the dimension of the node states, and σ is a non-linear function.

Finally, Veličković et al. [VCC+17] base their architecture on the self-attention mechanism, allowing for node representation learning based on the attention given to its neighbours. The backbone of this approach is the Graph Attention Layer (GAT). This architecture already employs an attention mechanism where they express how important the features of node j are for node i as [VCC+17]:

$$e_{ij} = a(W\vec{h}_i, W\vec{h}_j)$$

Here $W \in \mathbb{R}^{f'\times f}$ is a weight matrix that is used to project the node feature vectors $h_i, h_j \in \mathbb{R}^f$ into a higher-dimensional space $\mathbb{R}^{f'}$, then $a : \mathbb{R}^{f'} \times \mathbb{R}^f$ is the self-attention mechanism.

The performance of the different networks for various datasets and tasks has been explored by Dwidendi et al.[DJL+20] resulting in a comprehensive overview of GNNs and benchmarking. You et al. [YYL20] designed the GraphGym framework to allow for the comparison between the GNNs based on the design and task space. Given the large number of frameworks and architectures that have been proposed, they saw the need not only to benchmark them, but also to approach their evaluation from another perspective. This evaluation framework allows us to study the GNNs from a different perspective, not just based on the data domain or task, as previously done with the benchmarking.

To summarize, we briefly introduced the concept of GNNs and gave an overview of some of the variations and derived frameworks. The aim of this overview was to give the necessary content to the Graph transformer architecture, which is what we use in our work, since GNNs or variations thereof are still part of the Graph Transformer as described in the next section.

2.3 The Graph Transformer

Transformers [VSP+17] have seen great success in the NLP domain. Originally designed for machine translation, they leverage the encoder-decoder architecture to predict the next token in a sequence based only on the previous sequence. They do this by computing the attention weights, which say which parts of the sequence before are most important for the prediction of the next token in the sequence. This approach works well on textual data. In this section, we indeed show that the architecture is also applicable to relational data represented as graphs, in our case.

2.3.1 From the Transformer to the Graph Transformer

In the previous section, where we introduced GNNs, we already came across architectures that very closely resemble the core component of Transformers - the attention mechanism. Particularly, GATs already employ the attention mechanism, since the message-passing here is done using a local attention mechanism between the node and its local neighbourhood. Since GATs calculate the aggregated message from all the nodes' neighbors, resulting in one aggregated attention score, this can be interpreted as looking at the relationships between the nodes from different angles [Jos25].

Dwivedi and Besson [DB20] take the attention mechanism of the Transformer and adapt it specifically for the graph learning domain, introducing the Graph Transformer. The attention mechanism for a layer l in the Graph Transformer is defined as follows [DB20]:

$$\hat{h}_i^{\ell+1} = O_h^{\ell} \mathop{\Big\Vert}_{k=1} \Big(\sum_{j \in N(i)} w_{ij}^{k,\ell} V^{k,\ell} h_j^{\ell} \Big)$$

$$w_{ij}^{k,\ell} = \mathrm{softmax}_j \Big(\frac{Q^{k,\ell} h_i^{\ell} \cdot K^{k,\ell} h_j^{\ell}}{\sqrt{d_k}} \Big)$$

where we have the query, key and value matrices $Q_i^{\ell,k}, K_j^{\ell,k}, V_j^{\ell,k} \in \mathbb{R}^{d_k \times d}$, and the weight matrix of the projection $O_h^{\ell} \in \mathbb{R}^{d \times d}$. The result is then concatenated over the k attention heads.

The authors make two key observations that allow them to define this novel framework and outperform the GAT and GCN architectures. First, they leverage graph sparsity, arguing that although a graph is a globally interconnected structure with various sizes of nodes, the local connections are more desirable as the inductive bias and the global information can be incorporated differently in order to benefit from the feasibility of the local computations. This brings them to the second key concept: positional encodings. Just like the Transformer uses the sinusoidal positional embedding for each token of the input text, the graph structure always requires the positional information to be encoded. By now, this concept should be familiar, as it is exactly the part of the whole Transformer architecture that our work is focused on.

In the Graph Transformer, the authors opted for the Laplacian positional encoding stemming from the original Transformer positional encodings. They develop the framework for both the case when all the learning is based solely on the node embeddings, as well as allow for leveraging edge features to modify the attention scores between the corresponding node pair.

Finally, comparing the attention mechanism from the Graph Transformer and the GATS, the equivalence between the multi-head attention in Transformers and the message-passing layer in GATs is established, as Joshi [Jos25] puts it, *"transformers are GNNs over fully connected graph."*

2.3.2 Graph Transformer Architectures

The Graph Transformer introduced in the previous section is just one of many Transformer-based architectures for graph learning that have recently emerged.

Kreuzer et al. [KBH$^+$21] defined the Spectral Attention Network (SAN) architecture and addressed the shortcomings of the Graph Transformer proposed by Dwivedi and Besson [DB20]. They use observations from spectral graph theory to define the main key points when using the Laplacian eigenvalues and eigenvectors as positional encodings used in these architectures. The main points stemming from this work include the normalization of the chosen eigenvectors, the choice and ambiguity of the eigenvalues implemented into the model using linear combinations, and the sign ambiguity amended by flipping the signs of the pre-computed eigenvectors at random during training. Given that SAN is a fully connected Transformer architecture, it does not suffer from some common issues present in the message-passing GNNs. These include over-smoothing, which is an effect when the message computed in the current iteration overweighs what representation has been learned so far [Ham20]. Eventually, as more layers are processed, it becomes hard to distinguish between the representations [AY21]. Another phenomenon is over-squashing, which happens when the communication paths become constrained and the information does not reach between nodes with a long-range connection, which naturally does not occur in a fully connected transformer layer. Lastly, the authors show that SAN is able to learn physical representations such as molecular data thanks to a more efficient use of the Laplacian spectrum.

Ying et al. [YCL+21] proposed the Graphformer, which brings the novel spatial encoding integrated into the architecture. In their architecture, they include three different types of encodings to capture the graph structure: centrality, edge, and spatial encoding. The centrality encoding leverages the node degree centrality to capture the importance of the node, which is a common feature explored in graph modelling social networks, for example. The spatial encoding addresses the fact that although Transformers are fully connected and global attention is available to every node, graphs are not sequential as we discussed previously, and hence they introduce a function that, for each pair of nodes, computes their spatial relation, to be specific, the shortest path is used. Lastly, to leverage edge features, a learnable embedding is used, which averages over the dot products of the edge features on the shortest path between each node pair. Graphformer outperforms both the GT and SAN. The limitation remains in the quadratic complexity of the attention mechanism, which poses a challenge mainly for large graphs.

Wu et al. [WJW+21] focused on addressing the lack of well-performing GNNs on graph data with long-range interactions. In their GraphTrans architecture, they simply combine a block of GNN, followed by combining the outputs with node encodings and passing them to the Transformer block, which can compute interactions between all node pairs. In this approach, the limitations of the GNNs, such as over-smoothing and over-squashing, remain a concern, since the GNN layer might fail to capture the needed relationship, which is then input into the Transformer that will not be able to recover the lost information missing from this received input [RGD+22].

Finally, Rampášek et al. [RGD+22] design the GraphGPS modular framework, which allows for the customization of a more generic Graph Transformer architecture, including the choice of layers, encoding, networks, and the configurations of all these components. The architecture has three pillars: the positional/structural encoding, local message-passing algorithm (MPNN layer), and global attention mechanism (GlobalAttn layer). This way, each of these modules can be chosen and customized to build the version that best suits the needs. Moreover, the complexity is linear in the nodes and edges, making it suitable for larger graphs that the other transformers struggle to deliver. In the message-passing component, the authors define an MPNN-Transformer combined architecture, which makes up the GPS layer. This approach aims to alleviate the problems with over-smoothing, which we discussed in relation to the limitations of other architectures, such as GraphTrans, where the Transformer follows the GNN. In GraphGPS, each layer aggregates the outputs of the MPNN and GlobalAttn layers, combining the local and global information. Again, each component can be chosen independently, given that the MPNN layer is based on local message-passing algorithms and the GlobalAttn layer is a fully connected attention mechanism capable of capturing global interactions. The first component we left out of the description here, the positional/structural encoding, is described more in depth in the next section.

To summarize, inspired by the message-passing algorithm's success in GNNs in the graph learning field and the potential that the Transformer architecture offers for global connections, the Graph Transformer architecture was created. Across the different proposed architectures, the main idea remains the same: leverage the message-passing algorithm for aggregating information from node neighbourhoods and apply the attention mechanism of the Transformer to allow nodes to attend to all other nodes of the graph, regardless of their connectivity. We saw architectures that successfully dealt with the over-smoothing and over-squashing problems that the GNNs commonly faced. However, with the introduction of the Transformer, new challenges arose, given the quadratic complexity of this approach. On a final note, we discuss the positional/structural encoding of a graph separately in the upcoming section, as this is often the key component to learning graph representations [DB20, YCL+21, WJW+21, RGD+22], and is the main focus of our work.

2.4 Overview of Positional/Structural Graph Encodings

To give an overview of the different approaches to graph encodings, specifically in the context of integrating them in GTs, we use the classification proposed by Rampášek et al. [RGD+22]. The authors distinguish between two categories of graph encodings: positional and structural, which furthermore can be classified as local, global, and relative, resulting in six categories in total. The positional encodings are those that capture the distance between nodes, whereas the structural encodings focus on structure similarity. On the second level of the classification, the distinction is made based on which feature the encoding draws from. For node features, we have the local category, encoding based on edge features would be in the relative category, and graph features are grouped in the global encodings. In the next sections, we introduce only the relevant encodings that we will be using for the experiments in this thesis.

2.4.1 Random-Walk Positional Encoding (RWSE)

The Random-Walk Structural Encoding (RWSE) is a method used for extracting features from the graph based on a random-walk computation. It consists of a random-walk operator, defined as $RW = AD^{-1}$ [DLL+22], where $A \in \{0,1\}^{n\times n}$ is the adjacency matrix and $D \in \mathbb{N}^{n\times n}$ is the degree matrix of A, which is a diagonal matrix containing the degrees of each node along the main diagonal. Then, for a given k, which represents the resulting dimension of the encoding, and a node i, its encoding is defined as RW_{ii}^{k}. In each application of the random-walk operator, the move from node i to its neighbor j is done with the probability of $deg(i)$. After the k-th application, we look at the probability of the random walk ending in the source node after k steps. This gives us the idea of the position of this node with respect to the other nodes based on the graph traversal, choosing a random edge to traverse at each step.

We see that this encoding operates on the edge features, which are represented by the degree matrix D. Hence, it is an example from the relative class of encodings. It performs well on the molecular datasets [RGD+22] and achieves state-of-the-art performance on the Peptides dataset.

2.4.2 Laplacian Eigenvectors Positional Encoding (LapPE)

Laplacian Eigenvectors Positional Encoding (LapPE) is the encoding used by Dwivedi and Besson [DB20] in the Graph Transformer they introduced. They reason that the LapPE for graphs is what the sinusoidal encoding used in the Transformer is for text. It is defined as the factorization of the Laplacian [DLL+22]:

$$\Delta = I_n - D^{-1/2} A D^{-1/2} = U^\top \Lambda U$$

where $I_n \in \{0,1\}^{n \times n}$ is an identity matrix, $A \in \{0,1\}^{n \times n}$ is the adjacency matrix, $D \in \mathbb{N}^{n \times n}$ is the degree matrix, $\Lambda \in \mathbb{R}^{k \times k}$ is the eingenvalues matrix and $U \in \mathbb{R}^{k \times n}$ is the matrix made of the eigenvectors.

As discussed previously, the major limitation of this encoding is the sign invariance, which stems from the eigenvalues being sign-ambiguous. This results in the architecture having to learn this additional invariance at the cost of the performance [DJL+20]. This encoding is more suitable for the image pixel data according to [RGD+22] and is the current best-performing encoding for the CIFAR10 dataset.

2.4.3 Motif Structural Encoding (MoSE)

The Motif Structural Encoding (MoSE) [BJB+25] is based on homomorphism counts in graphs. The homomorphisms are counted on the node level and bring a strong improvement over the RWSE encoding, as the expressivity is no longer bounded in the WL hierarchy.

MoSE is the state-of-the-art encoding method on the ZINC molecular dataset.

CHAPTER 3

Factorization of Graph Adjacency Matrices

In the first part of the thesis, we apply the methods for low-rank matrix approximation to the binary adjacency matrices of the graphs and analyze how the factorization methods capture the graph structure. We measure the relative reconstruction error between the original adjacency matrix and its lower-dimensional encoding. Then, we study the differences in the resulting encodings between the datasets and gain insights into how the choice of the encoding dimensionality relates to the reconstruction error, as well as how the choice of the factorization influences the encoding. Finally, we propose a novel method, based on the LPCA called LPCA-sim, and define an additional metric for evaluating the low-dimensional encoding compared to the input adjacency matrix. Based on the analysis of the results, we compose a collection of pre-computed encodings to be integrated into the Graph Transformer, which is described in the following chapter. The code for all the experiments described in this chapter is available in github.com/teriolx/binary-matrix-factorization.

3.1 Datasets

Throughout our experiments, we work with the three datasets ZINC, Peptides-func, and CIFAR10 from the PyTorch Geometric library [FL19]. The datasets contain predefined train, validation, and test splits, allowing for straightforward integration into the training of the graph transformer models and for benchmarking performance metrics.

The ZINC dataset contains graphs of molecular data and has 10,000 graphs in the training set and 1,000 graphs in the validation and test set, which is the smaller collection also called ZINC-12k. Peptides-func is another molecular dataset, which contains a total of 15,535 graphs of peptides, split 70/15/15. Finally, CIFAR10 is an image classification dataset where pixels from images are aggregated into superpixels. It has 45,000 graphs in the train subset, 5,000 graphs in the validation subset, and 10,000 graphs in the test subset.

Each graph has the following attributes:

- `x` is the node feature matrix.
- `edge_attr` is the edge feature matrix.
- `y` is the target value, either
  - a floating point number for the regression task on ZINC,
  - an integer value representing the class label for multi-class classification on CIFAR10,
  - a vector indicating the labels for the multi-class classification on Peptides-func.
- `edge_index` represents the edges between nodes by storing the node pairs that have an edge between them in two arrays. The length of the arrays corresponds to the number of edges in the graph. In a graph with n nodes, the nodes are numbered 0 to $n-1$, which is then the index used to retrieve the features of each of the nodes from x.

In this part, we use the datasets to construct the adjacency matrix of each graph, which is then used as our input into the matrix factorization methods. For that, we only utilize the `edge_index` attribute and construct the adjacency matrix defined by it. The other graph attributes are not relevant in this section and are used only later in the transformers.

The adjacency matrix is constructed by creating an $n \times n$ matrix, where n is the number of nodes, and using the two arrays given by `edge_index` to determine which cells are set to 1, indicating there is an edge between the node pair defined by the corresponding row and column. As discussed in the introduction, this resulting adjacency matrix is not the sole representation of the graph, as there is no defined ordering of the nodes. However, for the sake of our experiments, we only construct this one representation and keep it fixed throughout the experiments.

To gain better insights into the structure of the graphs in the datasets, we explore the distribution of the number of nodes per graph per dataset. Figure 3.1 shows the corresponding histogram. We observe that ZINC is the dataset with the smallest graphs, averaging around 22 nodes per graph, and Peptides has the largest graph with up to 400 nodes per graph. Moreover, we note that the distribution in ZINC and CIFAR10 follows a rather normal distribution, whereas the distribution for Peptides is right-skewed.

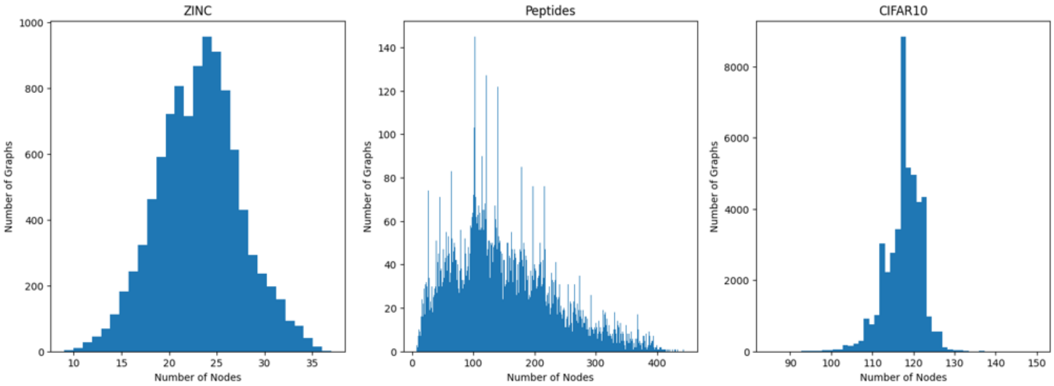

Figure 3.1: Histograms for number of nodes per graph in the datasets - ZINC, Peptides, CIFAR10.

3.2 Sign-Rank Approximation

Truncated Singular Value Decomposition (tSVD)

For the tSVD, we utilize the implementation provided in the Scikit-learn library [PVG+11]. Particularly, we use the tSVD class that allows us to specify the number of components, which is the dimensionality of the resulting matrices. Given that we decompose the input matrix into k components, we build the encoding as follows: the $L \in \mathbb{R}^{n\times k}$ matrix is directly the result of the transformation, meaning the results of `fit_transform`, and the $R \in \mathbb{R}^{k\times n}$ matrix is a matrix made up of the right singular vectors, accessible under the `component_` attribute of the fitted tSVD object. Finally, the encoding is the concatenation of the two matrices along their columns $\left[L \middle| R^\top\right] \in \mathbb{R}^{n\times 2k}$.

Logistic Principal Component Analysis (LPCA)

The code for LPCA is adapted from the Exact Embeddings project[1] based on [CMST20]. In their approach, the authors formulate the matrix factorization as an optimization problem by implementing the LPCA loss functions and factorizing the matrix using the minimizer from the SciPy library [VGO+20].

As input, the method requires the desired dimension k for the resulting matrices. Based on that, the factors are initialized uniformly at random with values from the interval $[-1, 1]$ given that the input adjacency matrix A is shifted to contain only the values -1 or 1, i.e., $\widetilde{A} = 2A - 1$. Then the initial factors are passed to the minimizer together with the loss function, and the optimization is run for a maximum of 2000 iterations with the `L-BFGS-B` optimization method [ZBLN97].

We also set the bounds for the values of the factors to $[-4, 4]$. The restriction on the values in the factorized matrices was necessary because, without specifying these intervals, the factorization sometimes resulted in very large ranges of values in the matrices. Then,

[1]https://github.com/schariya/exact-embeddings/blob/master/ExactEmbeddings.ipynb

once this encoding was integrated into the respective transformer model, the large values of the encoding compared to the node features in the input matrix led to much larger attention scores for the encoding, and the node features were not attended to due to their smaller scores compared to the encoding scores. Therefore, the bounds for the LPCA act as a type of input normalization, and the chosen bounds interval corresponds to the range of values of the encoded node features of the graphs.

Each iteration follows multiple steps. First, the factors are arranged in a way that they represent the two matrices, $L \in \mathbb{R}^{n \times k}$ and $R \in \mathbb{R}^{n \times k}$, which is done by splitting the factors array into two array of the same size and reshaping each half into the two matrices. With these matrices, the logits are computed as $W = LR^{\top}$.

Then the logarithmic probabilities are calculated for each element of the matrix:

$$p_{ij} = \sigma(-w_{ij} \cdot \tilde{a}_{ij})$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the logistic function, w_{ij} and $\tilde{a}_{ij}$ represent the elements in the i-th row and j-th column in the W and $\widetilde{A}$ matrices respectively.

With that, the loss which relaxes the non-convex property of $w_{ij} \neq \tilde{a}_{ij}$ is calculated using the following formula:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \log \left(1 + e^{-w_{ij}\tilde{a}_{ij}}\right)$$

Based on this, the two gradients for each of the matrices are defined as follows:

$$(\nabla L)_{ij} = -(p_{ij} \cdot \tilde{a}_{ij}) \cdot r_{ji}$$
$$(\nabla R)_{ij} = -l_{ji} \cdot (p_{ij} \cdot \tilde{a}_{ij})$$

Here $\tilde{a}_{ij}$, l_{ij} and r_{ij} are the elements in the i-th row and the j-th column of the $\widetilde{A}, L, R$ matrices respectively.

Finally, the calculated loss and the flattened gradients are passed to the next iteration. The iterations are repeated until convergence or until the maximum number of iterations is reached. We then construct our encoding by concatenating each row of the L matrix with each corresponding column of the R matrix and reconstruct the adjacency matrix $\widehat{A} = \text{round}_0(LR^{\top})$.

Lastly, the Frobenius error norm is calculated as $||\widehat{A} - A||_F$. The loss minimizes the Frobenius error norm between the adjacency matrix and its lower-dimensional representation [CMST20]. Therefore, when the error after the matrix factorization remains larger than zero, we conclude that the loss minimization failed to converge.

RQ1: Which of the two methods, tSVD or LPCA, produces a lower relative reconstruction error for a dimension k per dataset?

LPCA achieves significantly lower reconstruction error for the same k compared to tSVD for all datasets. Figure 3.2 shows the comparison between tSVD and LPCA errors on the ZINC dataset for various sizes of the encoding k. The results from LPCA at $k = 4$ already achieve very low error even for larger graphs, whereas tSVD for the same k is significantly worse with error much closer to the $k = 2$ case, whereas LPCA at $k = 4$ is already very close to the perfect reconstruction achieved with $k = 8$ using LPCA.

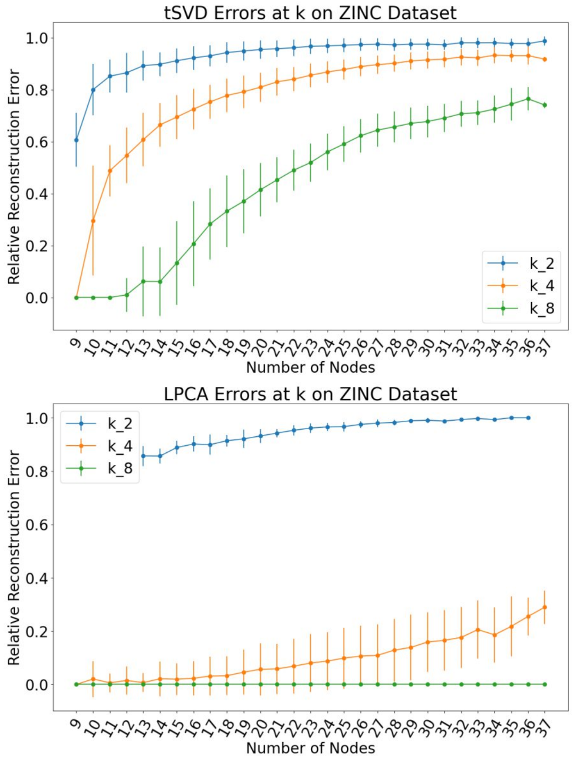

Figure 3.2: Relative reconstruction error vs. graph size for various encoding sizes k using tSVD (top) and LPCA (bottom).

RQ2: What dimension k is needed to achieve perfect reconstruction of the adjacency matrices for all of the graphs in each dataset?

By perfect reconstruction, we mean that the relative reconstruction error is zero, and we are looking for such k that provides such a reconstruction for each graph in the dataset. With tSVD, the encoding size k needed to achieve perfect reconstruction is linear in the size of the graph, which is depicted in Figure 3.3. For LPCA, this is not the case, and this method is able to achieve perfect reconstruction with k much smaller than the graph size. This results in the k being much higher on all datasets when using tSVD compared to LPCA, as Table 3.1 shows.

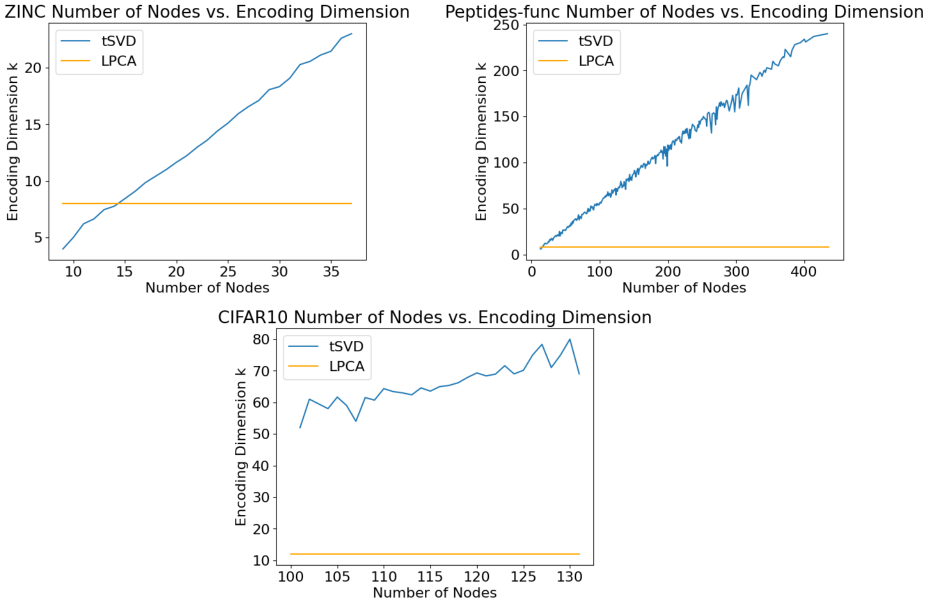

Figure 3.3: Comparison between tSVD and LPCA encoding dimensionality for matrices with zero relative reconstruction error.

method	ZINC	Peptides	CIFAR10
LPCA	8	8	12
tSVD	24	250	80

Table 3.1: Encoding dimension k needed to achieve perfect reconstruction per method and dataset.

Based on these findings, we set $k = 8$ for the remainder of the experiments on ZINC and Peptides-func, and $k = 12$ for the CIFAR10 dataset.

3.3 Node Encoding and Neighbourhood Distance

To analyze how well the graph encodings reflect the structural differences of the nodes in the graph, we propose the *neighbourhood distance* metric. Neighbourhood distance measures the difference in the neighbourhoods of two nodes in a graph. Consider the graph shown in Figure 3.4 and the two nodes colored in gray. These two nodes have identical neighbourhoods, meaning we are not able to distinguish between them from a structural perspective. On the other hand, the neighbourhoods of the grey node and the green node differ significantly. The neighbourhood distance metric allows us to measure this property for a graph $G = (V, E)$ and for each node pair $v, w \in V$ like so:

$$d(v, w) = |N(v) \triangle N(w)|,$$

where $N(x) \subseteq V$ is the neighbourhood of a node $x \in V$ and $\triangle$ is the symmetric set difference operator.

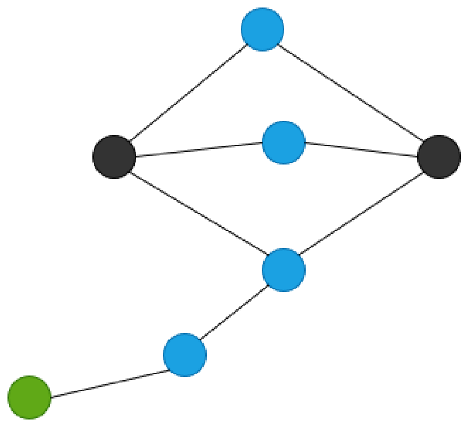

Figure 3.4: Illustration of structural differences between two nodes in a graph. The grey nodes have identical neighbourhood structures, whereas the green node and a grey node have very different neighbourhoods.

It is desired that the more similar two nodes in a graph are based on d, the more similar their encoding should be, because we aim to compute such an encoding that is able to distinguish between two node structures as best as possible. Essentially, the goal is that when looking at the encodings for a pair of nodes, we can tell whether they are structurally similar in the graph or not at all.

Let $e_v, e_w \in \mathbb{R}^{1\times 2k}$ be the encodings for nodes $v, w \in V$. The encoding is the concatenation of the corresponding row of the L matrix and column from the R matrix, i.e., $e_v = l_v | r_v$, where l_v, r_v are the v-th row of the $L, R^\top$ matrices, respectively. To measure the distance between a pair of node encodings, we take the norm of their difference:

$$s(e_v, e_w) = ||e_v - e_w||_2$$

By combining the encoding distance and the neighbourhood distance metrics, we define the graph encoding distance, which is computed using Algorithm 3.1. The algorithm computes the encoding distances of all node pairs in the graph and groups them by the node pair neighbourhood distances of the corresponding nodes.

Algorithm 3.1: Graph encoding distance

Input: Graph $G = (V, E)$, embeddings $\{e_v \mid v \in V\}$

Output: Map `distances` with arrays of the encoding distances with each distance d

foreach $v \in V(G)$ **do**
    foreach $w \in V(G)$ **do**
        $d \leftarrow d(v, w)$;
        if *`distances`[d] is undefined* **then**
            `distances`[d] $\leftarrow$ empty list;
        end
        Append $s(e_v, e_w)$ to `distances`[d];
    end
end

We apply the algorithm to each graph in each dataset and average the encoding distances after aggregating over d. This allows us to analyze the relationship between the neighbourhood distance and the average encoding distance. We refer to this relationship as the *graph encoding similarity*. In the following section, we compare the graph encoding similarity of encoding methods commonly used in the graph transformer models.

ZINC First, we inspect the distribution of the distances between each node pair across the dataset. Figure 3.5 shows that the distances follow a normal distribution.

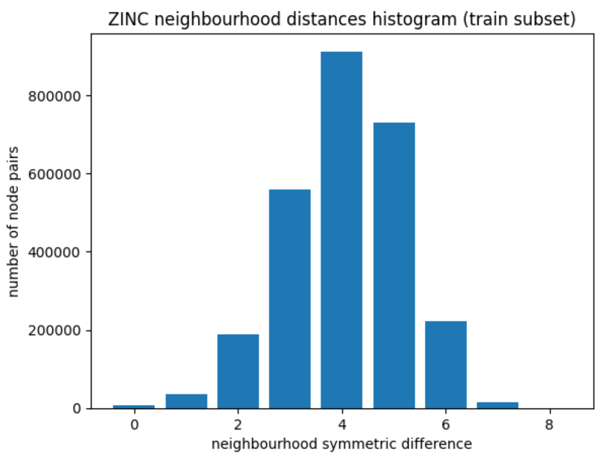

Figure 3.5: Neighbourhood distances distribution on the ZINC dataset.

Figure 3.6 shows the aggregated results of the calculated graph encoding similarity using RWSE and LPCA encodings. LPCA, compared to RWSE, has a much larger absolute value of the average distance between node encoding pairs, which indicates that the sizes of the encodings from LPCA are not normalized compared to RWSE, where the difference between two encodings is at most 1. Furthermore, the LPCA cannot effectively distinguish between nodes that have a neighbourhood symmetric difference larger than 4. On the other hand, RWSE has much more variance in the average distances at each point compared to LPCA. It is also important to note that the encodings here are of different sizes. LPCA has the size of 16, which is the result of stacking the two factorized matrices resulting from LPCA with $k = 8$, and RWSE is computed using 20 features.

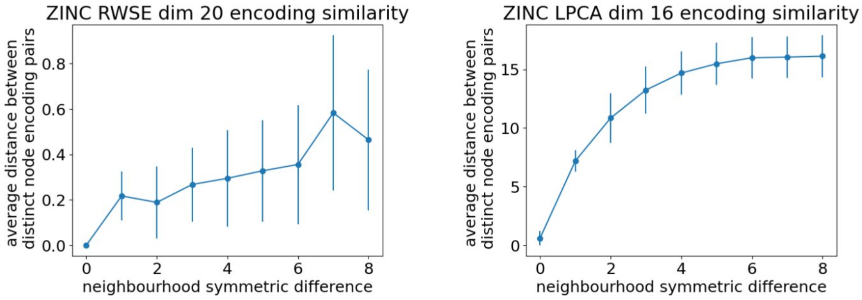

Figure 3.6: Graph encoding similarity of RWSE and LPCA on the ZINC dataset.

Peptides-func The distribution of the distances between each node pair in Peptides-func is very similar to the one in ZINC, as Figure 3.7 shows.

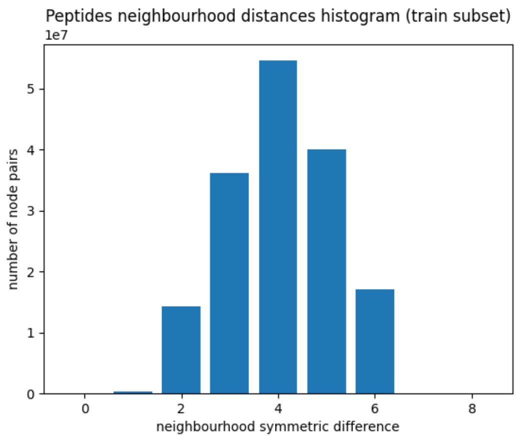

Figure 3.7: Neighbourhood similarity distances distribution on the Peptides-func dataset.

Figure 3.8 shows the aggregated results of the calculated graph encoding similarity using RWSE with 9 features, and LPCA with the encoding size of 16. The same observations as in the case of the ZINC dataset can be made. The RWSE has much smaller average distances, since the encodings from LPCA are not normalized, resulting in larger distances between them. Moreover, the LPCA struggles to distinguish between encodings with the neighborhood symmetric difference larger than 1, as the distances seem almost constant after this point and have a high variance as well.

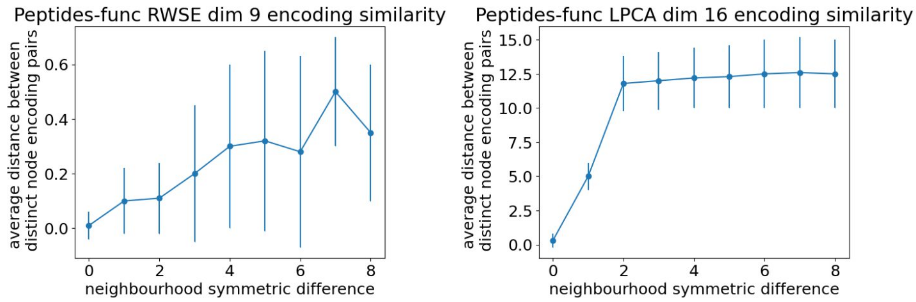

Figure 3.8: Graph encoding similarity of RWSE and LPCA on the Peptides-func dataset

CIFAR10 The distribution of the distances on the CIFAR10 dataset differs drastically from the molecular datasets due to the grid-like structure of the graphs in this dataset, resulting in a heavily left-skewed distribution of the distances shown in Figure 3.9.

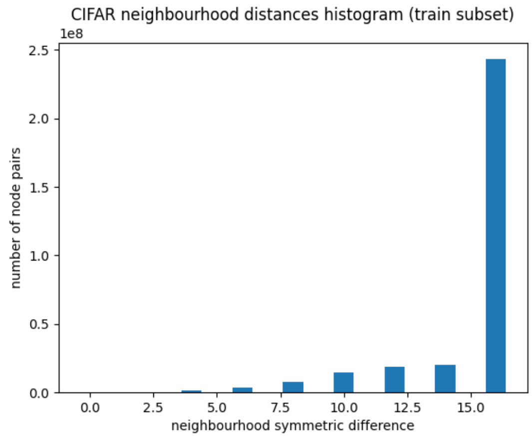

Figure 3.9: Neighbourhood similarity distances distribution on the CIFAR10 dataset.

Figure 3.10 shows the aggregated results of the calculated graph encoding similarity using RWSE with 17 features and LPCA with the encoding size of 24. The LPCA encoding is able to reflect the node neighbourhood similarity properly, where the closer the two node neighbourhoods are, the smaller the distance between the node encodings. RWSE on this dataset has a very low average distance between the encodings, but the node encoding pairs are hardly distinguishable due to the large variance, leading to all of the pairs looking alike based on the encoding distance despite their big neighbourhood symmetric difference. However, this is again likely due to the difference in the graph structure in this dataset rather than the factorization method itself.

When using the basic LPCA as the encoding, we obtain the graph encoding similarities that fail to distinguish well between the node encoding pair on the ZINC and Peptides-func datasets, as mentioned earlier. What we are looking for in these relationships is almost a linear curve, since the larger the neighborhood symmetric difference, the

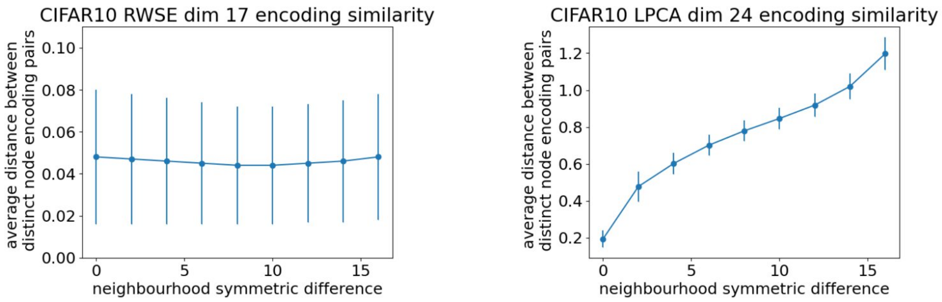

Figure 3.10: Graph encoding similarity of RWSE and LPCA on the CIFAR10 dataset.

larger the node encoding distance of that node pair should be to allow for distinguishing between node structures based on their encodings. However, the LPCA only shows the desired curve for CIFAR10. On the molecular datasets, it is hard to distinguish the nodes that have a symmetric difference greater than four from each other based only on their encoding. Ideally, we want the encoding to capture these vastly different node structures by encoding them significantly differently, which would be reflected in the encoding similarity decreasing.

3.4 LPCA with Graph Encoding Similarity (LPCA-sim)

We observe that the basic LPCA does not result in encodings that would capture the graph encoding similarity well. Therefore, we adjust the LPCA loss function by adding another loss term that optimizes the encoding similarity by using the neighbourhood and encoding distance metrics. We define the *LPCA-sim* method with the following loss function:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \log\left(1 + e^{-w_{ij}\cdot\tilde{a}_{ij}}\right) + \gamma \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{s(w_i, w_j)}{(d(i,j)+1)^2}$$

The first summand is the original LPCA loss, and the second summand represents the encoding similarity loss, where w_i, w_j are the i-th and j-th row of W corresponding to the encoding of nodes $i, j \in V$ at the given iteration, s is the encoding distance, and d is the neighbourhood distance. Since this loss introduces a trade-off between the reconstruction error loss from the basic LPCA and the encoding similarity, we introduce the γ hyperparameter to regulate the influence of the added loss term.

Additionally, the gradients are modified in the following way:

$$(\nabla L)_{ij} = -(p_{ij} \cdot \tilde{a}_{ij}) \cdot r_{ji} + \gamma \cdot \frac{2 \cdot (l_i - l_j)}{(d(i,j)+1)^2}$$
$$(\nabla R)_{ij} = -l_{ji} \cdot (p_{ij} \cdot \tilde{a}_{ij}) + \gamma \cdot \frac{2 \cdot (r_i - r_j)}{(d(i,j)+1)^2}$$

Here we take the i-th and j-th rows of the L and R matrices, which represent the node encodings.

One final modification that is introduced in the LPCA-sim method is the encoding normalization. It is desired that the encoding of each node has the same size, as the size of the encoding should not contribute to the structural representation of the node in the graph. Hence, we normalize the encoding obtained from the LPCA factorization by dividing the encoding vector e_i for a node $i \in V$ by the size of the vector:

$$\bar{e}_i = \frac{e_i}{\|e_i\|_2 + \varepsilon}.$$

Here, the encoding vector e_i is the concatenation of the corresponding rows from L and $R^\top$, i.e., $e_i = l_i | r_i$. We add a small regularization parameter $\varepsilon = 10^{-8}$ to prevent instability.

Finally, we compute the encodings with LPCA-sim and optimize for the similarity loss as well as low reconstruction error by tuning the γ hyperparameter. The gamma parameters were tuned so that the graph encoding similarity was captured better, while keeping the relative reconstruction error as low as possible.

CIFAR10 For the CIFAR10 dataset, we tune the hyperparameter on the whole training subset and obtain the reconstruction errors shown in Table 3.2. The corresponding encoding similarities are shown in Figure 3.11. We choose $\gamma = 2$ because the average distance in the node pairs is smaller in this case compared to $\gamma = 1$, and the encoding similarity curve is identical, so we are still capturing the same relationship.

	relative reconstruction error				
γ	85	100	115	130	145
1	0.13	0.13	0.18	0.24	0.31
2	0.41	0.54	0.66	0.76	0.85

Table 3.2: Relative reconstruction error per graph size with different γ parameters for the LPCA-sim encoding on the CIFAR10 dataset.

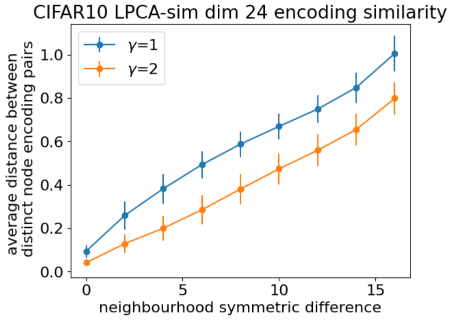

Figure 3.11: Graph encoding similarity on CIFAR10 using the LPCA-sim method with different γ values.

ZINC Table 3.3 shows the average relative reconstruction error for select graph sizes based on the different γ values tried. The evaluation was done on a subset of the training set using the first 1,000 graphs. The encoding similarity results are shown in Figure 3.12 for the different γ values. Based on the results, we chose $\gamma = 0.4$ for this dataset, as it provides an improvement in the captured encoding similarity and does not increase the error as much compared to the other values tried. It also has lower variance in the encoding similarity than the different values.

	relative reconstruction error				
γ	15	20	25	30	35
0.2	0	0	0.06	0.21	0.47
0.4	0.15	0.43	0.62	0.66	0.84
0.6	0.48	0.69	0.87	0.9	0.97
0.8	0.68	0.9	0.93	0.99	0.99

Table 3.3: Relative reconstruction error per graph size with different γ parameters for the LPCA-sim encoding on the ZINC dataset.

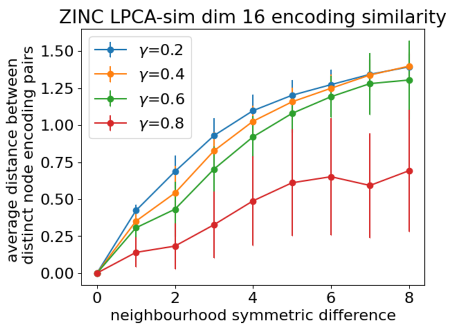

Figure 3.12: Graph encoding similarity on ZINC using the LPCA-sim method with different γ values.

Peptides-func For Peptides-func, the γ parameter increases the relative reconstruction error quite drastically, as shown in Table 3.4. The results were obtained from the first 50 graphs of the train set. Based on the calculated relative reconstruction error and the resulting encoding similarity shown in Figure 3.13, we opt for $\gamma = 0.1$ for this dataset. We make this choice because the larger γ values cannot capture the difference between the node pairs at all as the neighborhood symmetric difference increases.

	relative reconstruction error					
γ	33	70	100	155	225	365
0.1	0	0.6	0.85	0.92	1	1
0.2	0.36	0.94	1	1	1	1
0.3	0.85	1	1	1	1	1

Table 3.4: Relative reconstruction error per graph size with different γ parameters for the LPCA-sim encoding on the Peptides-func dataset.

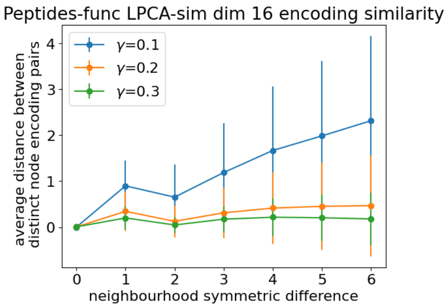

Figure 3.13: Graph encoding similarity on Peptides-func using the LPCA-sim method with different γ values.

We obtain the regularized encodings shown in Figure 3.14 for each of the datasets by computing the encodings with the chosen γ. We conclude that the structure given by the node neighbourhoods is now captured better than in the basic LPCA when using the neighbourhood similarity and encoding similarity as metrics.

3.5 Summary

In this chapter, we have explored two methods for factorizing binary matrices to obtain graph encodings from adjacency matrices. We saw that the LPCA achieves low error with very small dimensions relative to the graph size. We proposed an extension of the LPCA called LPCA-sim, together with a custom metric that reflects the relationship between the graph structure and its low-dimensional matrix representation. The proposed method focuses on incorporating the neighbourhood similarity information into the encoding and optimizes for both relative reconstruction error and encoding similarity. With LPCA-sim,

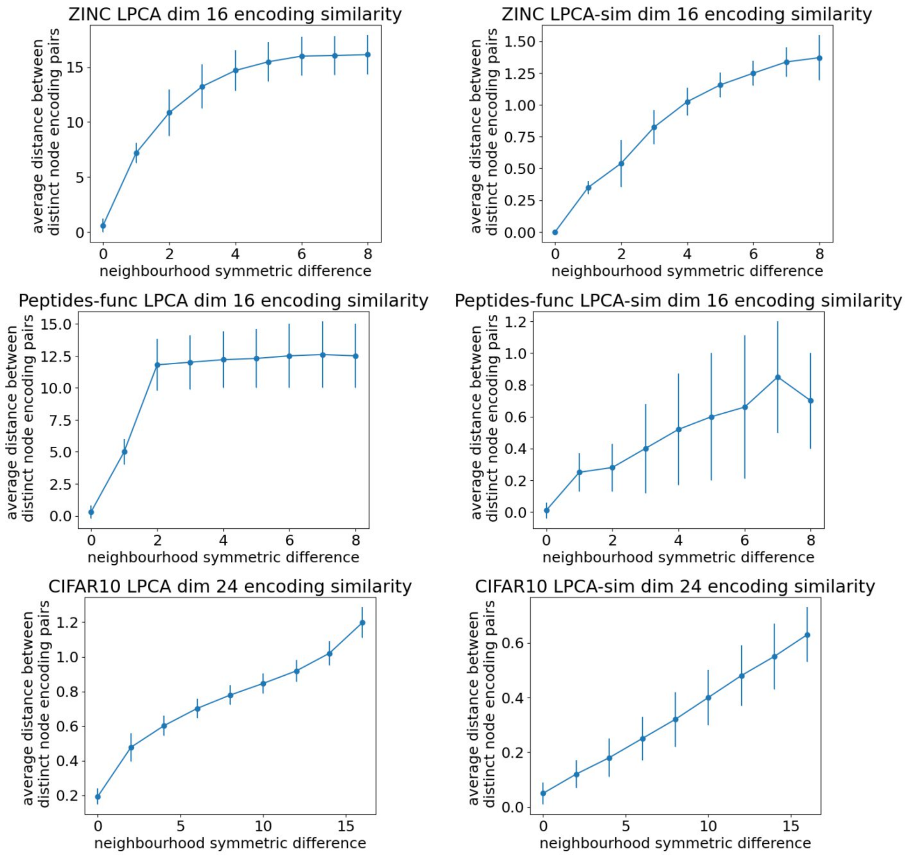

Figure 3.14: Graph encoding similarity for the LPCA-sim method (right) in comparison to LPCA (left) on the ZINC, Peptides-func, and CIFAR10 datasets.

we achieve a better relationship between the neighbourhood similarity distance and the encoding distance, but increase the relative reconstruction error.

CHAPTER 4

Graph Learning with Graph Transformers

In this chapter, we integrate the encoding obtained by applying the LPCA and LPCA-sim factorization methods into standard transformer models. We evaluate the performance and compare it with the performance of other commonly used encodings for the given task and dataset. The transformer models used in these experiments are built on top of the MoSE [BJB+25] project[1], which uses the GraphGPS [RGD+22] code base[2]. The code used in our experiments is available in the repository github.com/teriolx/graph-encoding-GT.

4.1 Integrating the Encoding into the Transformer

The way to integrate the encodings into the transformer is typically by concatenating them with the input feature matrix [DB20, RGD+22]. The input matrix is a $\mathbb{R}^{n\times m}$ matrix, where n corresponds to the number of nodes, and $m = j + 2k$ where j is the dimension of the node features encoding and $2k$ is the dimension of the LPCA positional encoding. Recall that the LPCA positional encoding is constructed as the concatenation of the factorized matrices $L, R \in \mathbb{R}^{n\times k}$, hence the width of the positional encoding is $2k$.

Formally, the input matrix is defined as follows:

$$\begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1j} & l_{11} & l_{12} & \cdots & l_{1k} & r_{11} & r_{12} & \cdots & r_{1k} \\ f_{21} & f_{22} & \cdots & f_{2j} & l_{21} & l_{22} & \cdots & l_{2k} & r_{21} & r_{22} & \cdots & r_{2k} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nj} & l_{n1} & l_{n2} & \cdots & l_{nk} & r_{n1} & r_{n2} & \cdots & r_{nk} \end{bmatrix}$$

[1] https://https://github.com/linusbao/MoSE
[2] https://github.com/rampasek/GraphGPS

Where each row i represents the node $i \in V$, the columns $f_{i1}, f_{i2}, ..., f_{ij}$ represent the encoded node features, $l_{i1}, l_{i2}, ..., l_{ik}$ are the rows of the L matrix and $r_{i1}, r_{i2}, ..., r_{ik}$ are the columns of the R matrix, or equivalently the rows of $R^\top$. The node feature encoding depends on the dataset and is described in more detail in the following section.

4.2 Experiments

We choose a different transformer model depending on the dataset, as different architectures are more suitable for different tasks and data. The aim is to observe how the new LPCA and LPCA-sim encodings perform in comparison to other commonly used encodings. We start from the same model configuration for all the encodings and then tune the hyperparameters as described only for the LPCA-sim method. All the results presented are averaged over multiple runs, and the standard deviation between the runs is also reported to allow for a more robust comparison.

4.2.1 CIFAR10

For the CIFAR10 dataset, we use the GPS model [RGD+22], and fix the following hyperparameters:

optimizer: adamW	batch size: 16	GNN activation: relu
weight decay: 0.0001	GT layers: 3	GNN dropout: 0.0
base learning rate (LR): 0.001	GT heads: 4	GNN aggregation: mean
max epochs: 100	GT inner dimension: 52	
LR scheduler: cosine	GT dropout: 0.2	
num warmup epochs: 5	GT attention dropout: 0.5	

We do not perform any hyperparameter tuning in this case, and all the encodings evaluated use the same model configuration. This only applies to the results computed in our experiments and not the MoSE and no encoding (none) performance metrics, which are taken from [BJB+25].

For each encoding, we average the results over four runs with varying seeds. LPCA-sim reaches comparable accuracy to LapPE and slightly worse accuracy than RWSE, as shown in Table 4.1.

4.2.2 Peptides-func

For the Peptides-func we also use the `GPSModel`. In this case, we take an approach using purely the transformer architecture with no local network. Therefore, the attention scores are only calculated based on the global attention in this case. All the hyperparameters can be found in the configs in the code repository.

The results in Table 4.2 are averaged over four runs with varying seeds. All of the encodings share the same initial configuration, whereas the hyperparameters for LPCA-

Encoding	Accuracy
RWSE	73.73 ± 0.32
LapPE	73.58 ± 0.14
LPCA-sim	73.57 ± 0.14
MoSE	73.50 ± 0.44
LPCA	71.58 ± 0.51
none	71.49 ± 0.19

Table 4.1: Classification accuracy on CIFAR10 using different encodings.

sim have been further tuned, and the results of the best hyperparameter configuration are presented in this case.

Encoding	Average Precision
RWSE	73.73 ± 0.32
LPCA-sim	52.27 ± 1.26
none	44.31 ± 1.07
LPCA	37.33 ± 1.42

Table 4.2: Average precision of the classification on Peptides-func using different encodings.

We start by fixing the following hyperparameters of the model:

optimizer: adamW	batch size: 128
weight decay: 0.0	GT dropout: 0.0
base learning rate (LR): 0.0003	GT attention dropout: 0.5
max epochs: 200	
LR scheduler: cosine	
num warmup epochs: 10	

In Table 4.3 we present the results of the hyperparameter tuning on the LPCA-sim encoding. We followed a local search approach to the hyperparameter tuning, starting from the initial configuration and focusing on trying different configurations for one hyperparameter at a time. In this process, we used three runs per hyperparameter configuration. We tuned the number of layers, heads, and inner dimension of the model.

As the results show, increasing the number of heads improved the performance, but increasing the inner dimension or the number of heads led to improvement only on the training set, but not on the test set. Increasing these parameters also leads to more model parameters to be learned, and since it brought no improvement, it is not necessary in this case to use the larger settings. Therefore, the configuration chosen as the best was the one with doubled layers compared to the initial configuration, but all other parameters were kept as the original ones.

Hyperparameter configuration	Train AP	Test AP
4 layers, 4 heads, 96 inner dimension	0.69	0.51
8 layers, 4 heads, 96 inner dimension	**0.75**	**0.54**
8 layers, 8 heads, 96 inner dimension	0.77	0.54
8 layers, 8 heads, 160 inner dimension	0.78	0.54

Table 4.3: Comparison of hyperparameter settings for the transformer model used in training on Peptides-func. The average precision is reported for each configuration on both the test and train sets.

4.2.3 ZINC

For ZINC, we use a custom implementation of a transformer model with edge features from [BJB+25]. This model uses layers that are purely based on the transformer attention mechanism without local attention. One unique feature introduced here is the full connectivity of all nodes through introducing the edge index as part of the attention calculation mask, where the edge features are also accounted for.

The node features here are encoded using an encoder that assigns a unique encoding to each type of node, which in this case corresponds to the atom type in the molecule graph.

Table 4.4 contains the errors averaged over three runs for each encoding, with the exception of MoSE, for which the results are taken directly from [BJB+25]. The runs across encodings share the same hyperparameter configuration, which is available in each corresponding configuration file in the code repository.

Encoding	Mean Absolute Error
MoSE	0.089 ± 0.018
RWSE	0.112 ± 0.019
LPCA-sim	0.196 ± 0.029
none	0.207 ± 0.025
LPCA	0.303 ± 0.039

Table 4.4: Mean absolute error of the regression on ZINC using different encodings.

The parameters that are fixed for this model are the following:

optimizer: adamW	batch size: 32
weight decay: 0.00001	GT layers: 10
base learning rate (LR): 0.001	GT heads: 4
max epochs: 2000	GT inner dimension: 64
LR scheduler: cosine	GNN activation: relu
num warmup epochs: 50	GNN aggregation: mean

We tune the dropout to try and combat the overfitting to the training set. Table **??** shows the results averaged over two runs. As the results show, the dropout does not aid in decreasing the MAE as it increases, hence we keep it at 0.

Dropout	Train MAE	Test MAE
0	**0.076**	**0.196**
0.05	0.162	0.207
0.1	0.111	0.228
0.15	0.118	0.296
0.2	0.132	0.297
0.25	0.191	0.298
0.3	0.446	0.446

Table 4.5: Comparison of dropout settings for the transformer model used in training on ZINC. The mean absolute error is reported for both the test and train sets.

Another configuration we have tried was adding extra normalization layers after the normalization layers that are already part of the model, but this did not bring an improvement to the MAE, which remained at 0.27 on the test set, averaged over two runs.

4.3 Discussion

First, we answer the third research question and its subquestions, based on the results from each dataset. Then we address additional research questions that emerged during the experiment process and discuss the obtained results.

RQ3a: How does the new encoding method compare to other encodings (RWSE, LapPE) in terms of accuracy for classification on CIFAR10?

On the CIFAR10 dataset, LPCA-sim matches the accuracy of LapPE, but does not outperform the RWSE.

RQ3b: How does the new encoding method compare to no encoding and RWSE in terms of average precision for classification on Peptides-func?

On the Peptides-func dataset, LPCA-sim performs significantly worse than the RWSE, but slightly better than having no positional encoding.

RQ3c: How does the new encoding method compare to no encoding and other encodings (RWSE, MoSE) in terms of mean absolute error for regression on ZINC?

On ZINC, the LPCA-sim encoding slightly improves the MAE over the no-encoding baseline, but does not match the performance of RWSE or MoSE.

How does LPCA-sim compare to LPCA?

We conclude that LPCA-sim outperforms LPCA on all the datasets which as depicted in Figure 4.1. This can be attributed to the added optimization for the graph encoding similarity that we introduced in LPCA-sim. The improvement of the results indicates that the intuition we had, that representing more similar nodes with more similar encodings and dissimilar nodes with encodings that are more distinguishable, yields better results.

However, the poor results we obtained when using the LPCA might be due to the lack of normalization in the LPCA encodings, which might lead to undesirably large attention scores at some nodes. These scores then signal that the nodes are more important, and their information gets amplified. However, since the node encoding size can vary based on the factorization result, the attention score might reflect the encoding size rather than the node importance. Such an effect is not desired because we aim to capture graph structure, and the size of the encodings is not reflective of that. Therefore, we also introduce encoding normalization in LPCA-sim to avoid such issues. In that sense, the improvement in the performance might also be attributed to this change in LPCA-sim compared to LPCA.

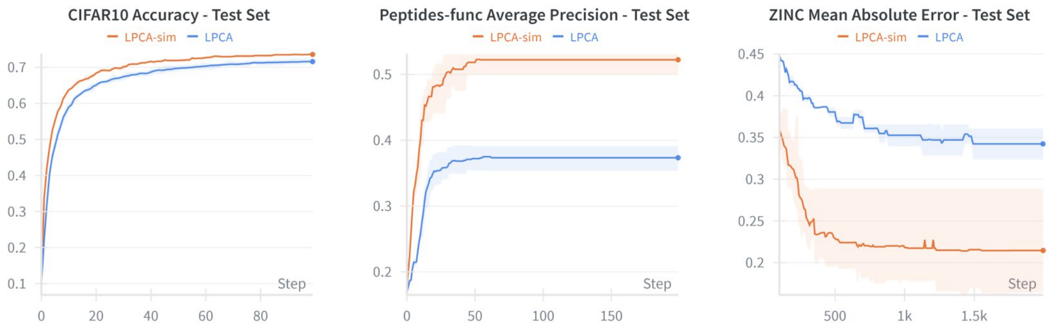

Figure 4.1: Comparison between LPCA and LPCA-sim encodings performance on the test sets. From left to right, accuracy of the classification on CIFAR10, average precision of the classification on Peptides-func, and mean absolute error of the regression on ZINC.

How does the performance of LPCA-sim on the training set compare to other encoding methods?

When training on the CIFAR10 dataset, the LPCA-sim encoding achieves 81.7 % accuracy on average, which is higher than both LapPE and RWSE, with a maximum of 77.6 % accuracy reached as depicted in Figure 4.2. As we have discussed previously, this increase in the performance on the training dataset does not translate well to the test set, meaning the model is learning the information from the training set well. Still, it is unable to generalize it to the test set.

However, this is not the case on the ZINC or Peptides-func datasets. Here, the performance on the training sets is still worse than the best-performing encoding methods. Particularly, on Peptides-func, LPCA-sim reaches the accuracy of 71.35 % on average compared to the average accuracy of 79.52 % reached by RWSE. On the ZINC dataset, the difference is smaller, but still the LPCA-sim has the mean absolute error of 0.074 on average compared to 0.065 when using RWSE.

To conclude, we see that LPCA-sim is able to provide information to the transformer models, so that they learn better than when using no encoding. However, the performance on the molecular datasets does not match the state-of-the-art results obtained from other encoding methods.

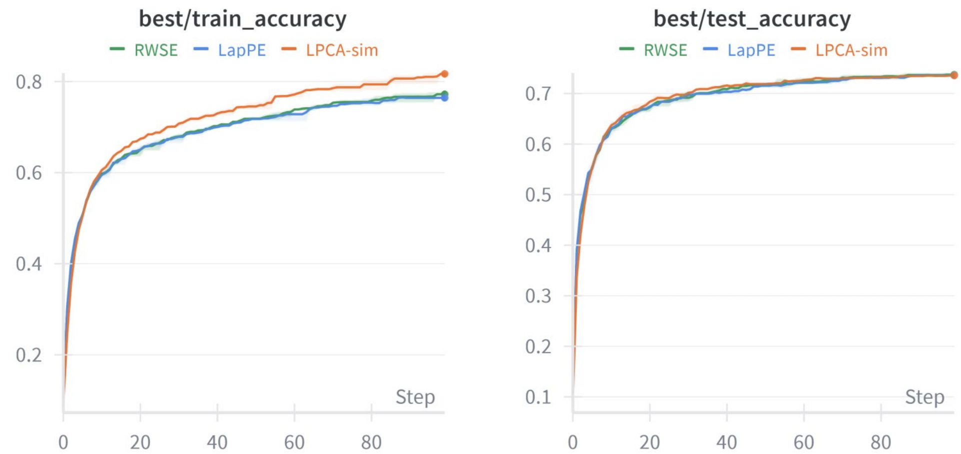

Figure 4.2: Accuracy of the classification on CIFAR10 for the train set (left) and test set (right) using RWSE, LapPE, and LPCA-sim encodings.

4.4 Summary

In conclusion, we showed that the LPCA-sim performs significantly better compared to the basic LPCA. This is mainly due to the normalization of the encodings that we added in LPCA-sim, as well as the additional information that we encoded based on the neighbourhood distance metrics.

We achieve competitive performance only on the CIFAR10 dataset, which leads to the conclusion that the LPCA-sim is able to capture the grid-like structure of the graphs in this dataset well. However, on the molecular datasets, the novel methods do not match the performance of the state-of-the-art methods and improve only slightly compared to the baseline of using no positional encoding at all. Based on that, we conclude that the LPCA-sim does capture the graph structure in the molecular datasets to some extent, which is useful for learning with transformers, but encoding the graph structure in a way that would significantly improve the graph learning still remains a challenge when using the factorization methods for encoding.

CHAPTER 5

Conclusion and Future Work

5.1 Contribution

In this thesis, we explored a novel approach to encoding graph structure using binary matrix factorization methods. We evaluated the encodings resulting from the tSVD and LPCA methods and achieved significant dimensionality reduction with the LPCA method. Based on that, we further modified the basic LPCA and proposed a novel method for factorizing adjacency matrices, LPCA-sim. With this method, we also introduce novel metrics for measuring the distance between two node encodings and the neighbourhood distance metric for measuring the distance between two nodes in the graph structure. Finally, by combining the two metrics, we define the graph encoding similarity as the relationship between them and evaluate our new encodings as well as other commonly used encodings using this metric.

With the computed encodings, we modified the graph transformer models to make use of the node positional encoding as part of the input and evaluated the performance compared to other encoding methods. The performance matches state-of-the-art methods only on the CIFAR10 dataset.

Based on our experiments, we conclude that the LPCA can factorize the adjacency matrices with much lower relative reconstruction error than the tSVD for the same encoding size. Moreover, LPCA requires a very small dimensionality compared to the tSVD to capture the adjacency matrix with no error. However, optimizing for the lowest reconstruction error has proven not to be the most performant encoding once integrated into the graph transformer. Therefore, with the introduction of the LPCA-sim method, we leverage the neighbourhood distance metric to optimize for node neighbourhood and encoding similarity, significantly improving the performance of the basic LPCA encodings.

5.2 Limitations

Although the encodings showed promising performance on the molecular datasets for the training splits, this performance was not reflected in the test set evaluation. Therefore, the challenge remains in the overfitting of the transformer to the training data, which then fails to generalize. A possible improvement in this case would be further tuning of the model and the encodings themselves to see whether a different configuration could provide better performance even on the test set.

Furthermore, since the encoding is comprised of the concatenation of the rows of the L and the columns of the R matrices resulting from the factorization, it is required to multiply the row by the column to reconstruct the adjacency. Such multiplication has proven to be a challenge for the attention mechanism of the graph transformer. One approach that could address this limitation is to factorize the adjacency matrix in such a way that results in one matrix only, for example, by fixing the L matrix at the start for all graphs in the dataset and only optimizing R. Such encoding would then correspond to the features represented as columns, which is how RWSE is also constructed and shows better results.

Another possible path to explore would be the attention mechanism itself, which was briefly explored in the work by adding a bias term to the attention, which was the value from the reconstructed adjacency matrix, but was left unexplored in the end. Such modifications to the attention mechanism might contribute to the

Lastly, the GraphGPS framework for the transformer models offers a solid base for training the transformer models, but it is rather complex in its modular approach, which also made the debugging process slightly more challenging. Defining own components to integrate into the architecture was necessary but not always completely straightforward, as it was a challenge to trace all the different components that are used at once. Another challenge when using this framework is the lack of transparency of all the hyperparameters that can be set for all the different modules that the model consists of. The majority of the parameters are set implicitly with default values. The values can be overwritten by passing new values in a configuration file for each run. However, listing all the model hyperparameters would be very tedious. Hence, the configuration files contain only those parameters for which the values will be overwritten. Therefore, the configuration file does not reflect the entire setup for any given run.

Also, the larger number of configurable parameters that go into this architecture was becoming hard to manage with a lot of parameters set implicitly and then additionally modified through config files for each run.

5.3 Future Work

The approach explored in this thesis opens the possibility of using other binary factorization methods as a new approach to encoding graph structure. We have shown that these methods can capture graph structure, although they do not achieve state-of-the-art results thus far. But this approach demonstrates that the direction is promising and worth exploring, perhaps with other methods of the same family or by addressing some of the limitations of the LPCA-sim method.

One main factor to address in future work is the lack of systematic analysis of how the different settings of the LPCA-sim impact the performance of the models. In this work, we focused first on choosing the encoding dimension based on the relative reconstruction error and later the graph encoding similarity, and then used the best encodings based on these metrics. It would be valuable to evaluate the different encoding dimensions by training the models on them to see whether our metrics of evaluating the encodings actually reflect how the model performs with these encodings. The same principle could be applied to the other variables in the process, such as the comparison between LPCA and LPCA-sim, but without the encoding normalization to further isolate the impact each change has on the encoding's performance.

Overview of Generative AI Tools Used

The following prompts were sent to ChatGPT [Ope25], the GPT-4o model, to aid in writing markdown in LaTeX. The results were never used without first checking for correctness. They were also further augmented to match the desired format exactly.

Prompt: Please write the LaTeX code for this mathematical formula, [picture with formula].

Prompt: Please give me the LaTeX code for a table with the following data, [table data].

Prompt: Can you rewrite this using matrix notation as LaTeX code, [expression].
Example: `(np.logaddexp(0,-logits*adj_s)).sum()`

Additionally, DeepL [Dee25] was used to aid in translating the abstract from English to German.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AY21] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[BJB+25] Linus Bao, Emily Jin, Michael M. Bronstein, İsmail İlkan Ceylan, and Matthias Lanzinger. Homomorphism counts as structural encodings for graph learning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[CDS01] Michael Collins, Sanjoy Dasgupta, and Robert E. Schapire. A generalization of principal components analysis to the exponential family. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 617–624. MIT Press, 2001.

[CMST20] Sudhanshu Chanpuriya, Cameron Musco, Konstantinos Sotiropoulos, and Charalampos E. Tsourakakis. Node embeddings and exact low-rank representations of complex networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014.

[DB20] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *CoRR*, abs/2012.09699, 2020.

[Dee25] DeepL SE. Deepl translator, 2025. Accessed: 2025-08-26.

[DJL+20] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *CoRR*, abs/2003.00982, 2020.

[DLL+22] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations, 2022.

[DRG+22] Vijay Prakash Dwivedi, Ladislav Rampásek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[FL19] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019.

[GDH+16] Rafael Gómez-Bombarelli, David Duvenaud, José Miguel Hernández-Lobato, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *CoRR*, abs/1610.02415, 2016.

[GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

[Ham20] William L. Hamilton. *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020.

[Han87] Per Christian Hansen. The truncatedsvd as a method for regularization. *BIT*, 27(4):534–553, December 1987.

[HMT11] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, 2011.

[HVG09] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *CoRR*, abs/0912.3848, 2009.

[HYL17] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017.

[Jos25] Chaitanya K. Joshi. Transformers are graph neural networks. *CoRR*, abs/2506.22084, 2025.

[KBH+21] Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 21618–21629, 2021.

[KW16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[LTBZ16] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[NGM16] Stefan Neumann, Rainer Gemulla, and Pauli Miettinen. What you will gain by rounding: Theory and algorithms for rounding rank. In Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 380–389. IEEE Computer Society, 2016.

[Ope25] OpenAI. Chatgpt (july 2025 version). Large language model, 2025. Accessed: 2025-08-26.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RGD+22] Ladislav Rampásek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing*

Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, 2022.

[SGT+09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

[SI15] Teague Sterling and John J. Irwin. Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015. PMID: 26479676.

[SSU03] Andrew I. Schein, Lawrence K. Saul, and Lyle H. Ungar. A generalized linear model for principal component analysis of binary data. In Christopher M. Bishop and Brendan J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, AISTATS 2003, Key West, Florida, USA, January 3-6, 2003*, pages 240–247. Society for Artificial Intelligence and Statistics, 2003.

[VCC+17] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *CoRR*, abs/1710.10903, 2017.

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[WJW+21] Zhanghao Wu, Paras Jain, Matthew A. Wright, Azalia Mirhoseini, Joseph E. Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference* *on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 13266–13279, 2021.

[YCL+21] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 28877–28888, 2021.

[YYL20] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[ZBLN97] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, December 1997.