Enhancing Graph Neural Networks with Spectral Graph Theory

Devin Kreuzer, School of Computer Science

McGill University, Montreal

March, 2022

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Computer Science

Preface

The following is a Manuscript-Based Thesis based on the paper entitled *Rethinking Graph Transformers with Spectral Attention* [32] published as a conference proceedings at *Advances in Neural Information Processing Systems 34* (NeurIPS 2021). A *Background* section was added to make the Thesis stand as an integrated whole.

Abstract

In recent years, the Transformer architecture has proven to be very successful in sequence processing, but its application to other data structures, such as graphs, has remained limited due to the difficulty of properly defining positions. Here, we present the *Spectral Attention Network* (SAN), which uses a learned positional encoding (LPE) that can take advantage of the full Laplacian spectrum to learn the position of each node in a given graph. This LPE is then added to the node features of the graph and passed to a fully-connected Transformer. By leveraging the full spectrum of the Laplacian, our model is theoretically powerful in distinguishing graphs, and can better detect similar sub-structures from their resonance. Further, by fully connecting the graph, the Transformer does not suffer from over-squashing, an information bottleneck of most GNNs, and enables better modeling of physical phenomenons such as heat transfer and electric interaction. When tested empirically on a set of 4 standard datasets, our model performs on par or better than state-of-the-art GNNs, and outperforms any attention-based model by a wide margin, becoming the first fully-connected architecture to perform well on graph benchmarks.

Abrégé

Ces dernières années, l'architecture Transformer s'est avérée très efficace dans le traitement des séquences, mais son application à d'autres structures de données, telles que les graphes, est restée limitée en raison de la difficulté de définir correctement les positions. Ici, nous présentons le *Spectral Attention Network* (SAN), qui utilise un codage positionnel appris (LPE) qui peut utiliser le spectre laplacien complet pour apprendre la position de chaque nœud dans un graphe donné. Ce LPE est ensuite ajouté aux caractéristique de nœud du graphique et transmis à un Transformeur entièrement connecté. En utilisant le spectre complet du Laplacien, notre modèle est théoriquement puissant pour distinguer les graphes et peut mieux détecter des sous-structures similaires à partir de leur résonance. De plus, en connectant entièrement le graphique, le Transformeur ne souffre pas de sur-écrasement, un goulot d'étranglement d'informations de la plupart des GNN, et permet une meilleure modélisation des phénomènes physiques tels que le transfert de chaleur et l'interaction électrique. Lorsqu'il est testé empiriquement sur un ensemble de 4 ensembles de données standard, notre modèle fonctionne au même niveau ou mieux que les GNN de pointe et surpasse de loin tout modèle basé sur l'attention, devenant ainsi la première architecture entièrement connectée à bien fonctionner sur des repères graphiques.

Acknowledgements

First and foremost, I would like to acknowledge and thank my advisor William L. Hamilton and co-author Dominique Beaini, for their countless support, continuous mentorship and important contributions to this entire process. Without their guidance, this work would have never been achievable.

I would also like to thank the remaining co-authors Prudencio Tossou and Vincent Letourneau for their important contributions.

Finally, I would like to thank my friends and family who supported me through the pandemic and enabled this entire research project.

I will forever be tremendously grateful to you all.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The prevailing strategy for graph neural networks (GNNs) has been to directly encode graph structure structure through a sparse message-passing process [21, 23]. In this approach, vector messages are iteratively passed between nodes that are connected in the graph. Multiple instantiations of this message-passing paradigm have been proposed, differing in the architectural details of the message-passing apparatus (see [23] for a review).

However, there is a growing recognition that the message-passing paradigm has inherent limitations. The expressive power of message passing appears inexorably bounded by the Weisfeiler-Lehman isomorphism hierarchy [37,39,53]. Message-passing GNNs are known to suffer from pathologies, such as *oversmoothing*, due to their repeated aggregation of local information [23], and *over-squashing*, due to the exponential blow-up in computation paths as the model depth increases [1].

As a result, there is a growing interest in deep learning techniques that encode graph structure as a *soft inductive bias*, rather than as a hard-coded aspect of message passing [18,29]. A central issue with the message-passing paradigm is that input graph structure is encoded by restricting the structure of the model's computation graph, inherently limiting its flexibility. This reminds us of how early recurrent neural networks (RNNs)

encoded sequential structure via their computation graph—a strategy that leads to well-known pathologies such as the inability to model long-range dependencies [25].

There is a growing trend across deep learning towards more flexible architectures, which avoid strict and structural inductive biases. Most notably, the exceptionally successful Transformer architecture removes any structural inductive bias by encoding the structure via soft inductive biases, such as positional encodings [48]. In the context of GNNs, the self-attention mechanism of a Transformer can be viewed as passing messages between all nodes, regardless of the input graph connectivity.

Prior work has proposed to use attention in GNNs in different ways. First, the GAT model [49] proposed local attention on pairs of nodes that allows a learnable convolutional kernel. The GTN work [58] has improved on the GAT for node and link predictions while keeping a similar architecture, while other message-passing approaches have used enhancing spectral features [10, 15] . More recently, the GT model [18] was proposed as a generalization of Transformers to graphs, where they experimented with sparse and full graph attention while providing low-frequency eigenvectors of the Laplacian as positional encodings.

In this work, we offer a principled investigation of how Transformer architectures can be applied in graph representation learning. **Our primary contribution** is the development of novel and powerful learnable positional encoding methods, which are rooted in spectral graph theory. Our positional encoding technique — and the resulting *spectral attention network (SAN)* architecture — addresses key theoretical limitations in prior graph Transformer work [18] and provably exceeds the expressive power of standard message-passing GNNs. We show that full Transformer-style attention provides consistent empirical gains compared to an equivalent sparse message-passing model, and we demonstrate that our SAN architecture is competitive with or exceeding the state-of-the-art on several well-known graph benchmarks.

Chapter 2

Background

2.1 Graphs

2.1.1 Basic Principles

Graphs are a ubiquitous form of data used to represent complex systems. They are used to encode a set of points with *nodes* and their connectedness with *edges*. As a result, the true power of graphs relies in their ability to demonstrate the relationships between points, as opposed to the sole properties of individual ones. See Figure 2.1 for example of a graph.

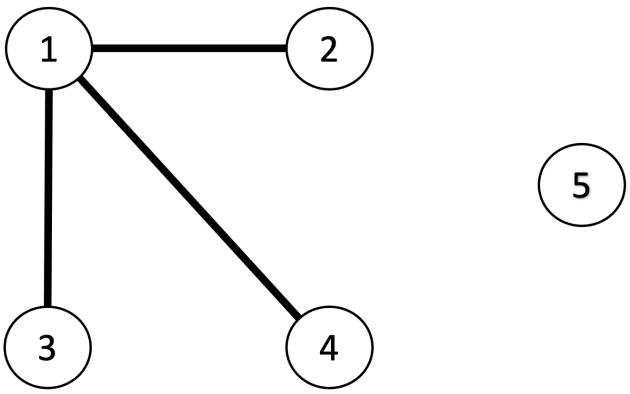


Figure 2.1: Example of a graph where the numbered points and lines represent nodes and edges respectively. It is possible for nodes to be disconnected from all others, as shown here.

Examples of the use of graphs include their ability to represent social networks, where nodes represents individuals and edges represent whether or not two individuals are friends. Further, in the molecular domain, we can represent molecules as graphs where individual atoms and their bonds are represented as nodes and edges respectively.

More formally, a graph $G = (V, E)$ is a tuple of a set of nodes V and edges E, where E is its own set of tuples $(u_1, u_2) \in E$, and $u_1, u_2 \in V$. In the context of machine learning on graphs, we are typically only concerned with simple graphs where there are no edges between a node and itself, at most one edge lies between any two nodes and edges are only unweighted and undirected: $(u_1, u_2) \in E \iff (u_2, u_1) \in E$.

Graphs are typically represented using an *adjacency matrix* $\mathbf{A}$, where we provide an arbitrary ordering of the nodes (as the one shown in 2.1) such that each column and row of the matrix indexes to a particular node (this arbitrary component has consequences we will revisit later). For the simple graphs we concern ourselves with, we can then encode the presence of an edge by setting $A[u_1, u_2] = A[u_2, u_1] = 1$, and filling the remaining entries with 0.

Finally, there are often certain attributes or properties associated with the elements of a graph. For example, in a molecular graph, we are often interested in the type of atom and bond associated with each node and edge respectively. We may even be interested in features at the molecular-level, such as the polarity of the molecule itself. Thus, along with each graph's adjacency matrix, we often have real-valued node-level and edge-level features matrices $\mathbf{X}_{node} \in \mathbb{R}^{|V| \times m}$, $\mathbf{X}_{edge} \in \mathbb{R}^{|E| \times n}$, and graph-level feature vectors $\mathbf{x}_{graph} \in \mathbb{R}^k$, where m, n, k denote the number of node, edge and graph features respectively.

2.1.2 Graph Laplacians

Although the adjacency matrix is an invaluable tool for representing the graph, the *Laplacian* is of equal fundamental importance. The unnormalized Laplacian is defined as follows [23]:

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{2.1}$$

where $\mathbf{A}$ is the corresponding adjacency matrix and $\mathbf{D}$ is the corresponding degree matrix; a diagonal matrix whose entries are given by the degree (number of connected edges) of the indexed node. The Laplacian matrix holds the following key properties [23]:

1. $\mathbf{L}$ is symmetric: $\mathbf{L} = \mathbf{L^T}$
2. $\mathbf{L}$ is positive semi-definite: $\mathbf{x^T L x} \geq 0, \forall \mathbf{x} \in \mathbb{R}^{|V|}$
3. $\mathbf{L}$ possesses the following identity: $\mathbf{x^T L x} = \sum_{(u_1,u_2)\in E}(\mathbf{x}[u_1] - \mathbf{x}[u_2])^2$
4. $\mathbf{L}$ has $|V|$ non-negative eigenvalues: $0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{|V|-1}$

Consider the eigendecomposition of the Laplacian $\mathbf{L}$:

$$\mathbf{L} = \mathbf{U \Lambda U^T} \tag{2.2}$$

where $\mathbf{U}$ is the corresponding matrix of eigenvectors and $\mathbf{\Lambda}$ the corresponding diagonal matrix of eigenvalues (interchangeably referred to as *frequencies* for reasons I will come back to). Since there is a natural ordering of the eigenvectors given by their eigenvalues (property 4.), we typically use the graph's *spectrum of eigenvalues* to denote a notion of 'high' and 'low' eigenvectors. For example, we call the eigenvector ϕ_0 associated to the lowest eigenvalue $\lambda_0 = 0$ the 'lowest' or 'smallest' eigenvector of the graph. As a result, we also usually form the matrix $\mathbf{U}$ according to this ordering. See 2.2 for visualization.

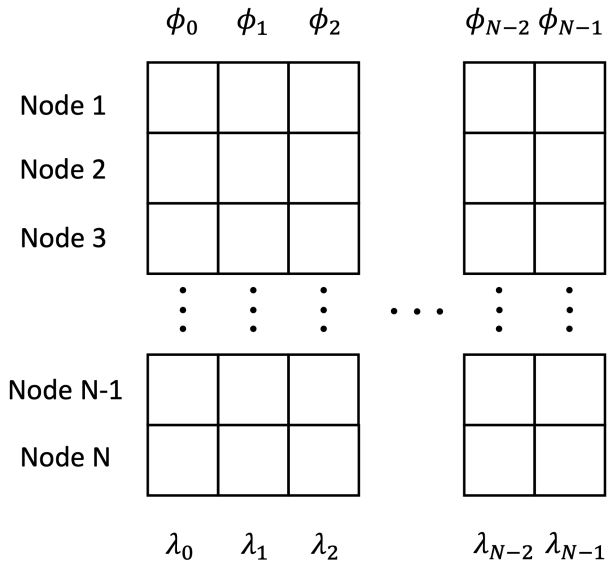


Figure 2.2: It is helpful to visualize the matrix of Laplace eigenvectors $\mathbf{U}$ as the ordered column-wise concatenation of eigenvectors based on their eigenvalues, where each row corresponds to a node in the graph.

2.1.3 Laplacian Eigenvectors

In order to understand the significance of the Laplacian eigenvectors and their eigenvalues, we must understand the connection between the Laplacian and the Laplace operator Δ used ubiquitously across the fields of mathematics and physics. The operator Δ in the context of arbitrary smooth functions $f : \mathbb{R}^d \to \mathbb{R}$ is used to compute the divergence ∇ of the gradient $\nabla f(\mathbf{x})$:

$$\nabla f(\mathbf{x}) = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x^2}$$

In basic terms, it is used to measure the average difference between the function f at the point $\mathbf{x}$ and the function at the points in the neighborhood of $\mathbf{x}$.

One can equally consider the Laplacian $\mathbf{L}$ as an operator on functions h that map the nodes of a graph to real values $h : V \to \mathbb{R}^{|V|}$. Let h be such a function and G be a graph with nodes V such that $h(V) = \mathbf{x}$:

$$(\mathbf{Lx})[i] = \sum_{j \in V} \mathbf{A}[i,j](\mathbf{x}[i] - \mathbf{x}[j])$$

This can be re-written as:

$$(\mathbf{Lx})[i] = \sum_{j \in N_i} (\mathbf{x}[i] - \mathbf{x}[j]) \tag{2.3}$$

where N_i is the set of nodes the node i is connected to through an edge (known as its *neighborhood*). Thus, the Laplacian $\mathbf{L}$ applied to functions over graphs has a similar intuition: applying it measures the average difference of a node i's functional value $\mathbf{x}[i]$ to those in its neighborhood.

Now, consider the following quantity (see properties 2. and 3. from Section 2.1.2):

$$\mathbf{x^T Lx} = \sum_{(i,j) \in E} (\mathbf{x}[i] - \mathbf{x}[j])^2 = C \geq 0 \tag{2.4}$$

This quantity can be thought of as representing the 'smoothness' of $\mathbf{x}$: low or high values of C mean that, on average, connected nodes have close or distant values in $\mathbf{x}$, respectively.

In particular, the problem of minimizing the quantity C is analogous to finding the eigenvectors and eigenvalues of the Laplacian $\mathbf{L}$ where $\boldsymbol{\phi} = \mathbf{x}$ and $\lambda = C$ by the Rayleigh-Ritz Theorem [23]:

$$\min_{\mathbf{x_i} \in \mathbb{R}^{|V|} : \mathbf{x_i} \perp \mathbf{x_j} \forall j < i} \frac{\mathbf{x_i^T L x_i}}{\mathbf{x_i^T x_j}}$$

Thus, the eigenvectors $\boldsymbol{\phi}$ correspond to assignments of values $\mathbf{x}$ which minimize 2.4 with quantity equal to the eigenvalue λ . In fact, the trivial solution $\boldsymbol{\phi_0}$ simply assigns a constant to all neighboring nodes to achieve a minimized value of $C = \lambda_0 = 0$. The second solution $\boldsymbol{\phi_1}$ is the orthogonal vector to $\boldsymbol{\phi_0}$ which next best minimizes the quantity $C = \lambda_1$. This process can be continued to find all the eigenvectors and eigenvalues of the Laplacian. Note that since the Laplacian can be thought of as a matrix but also as an

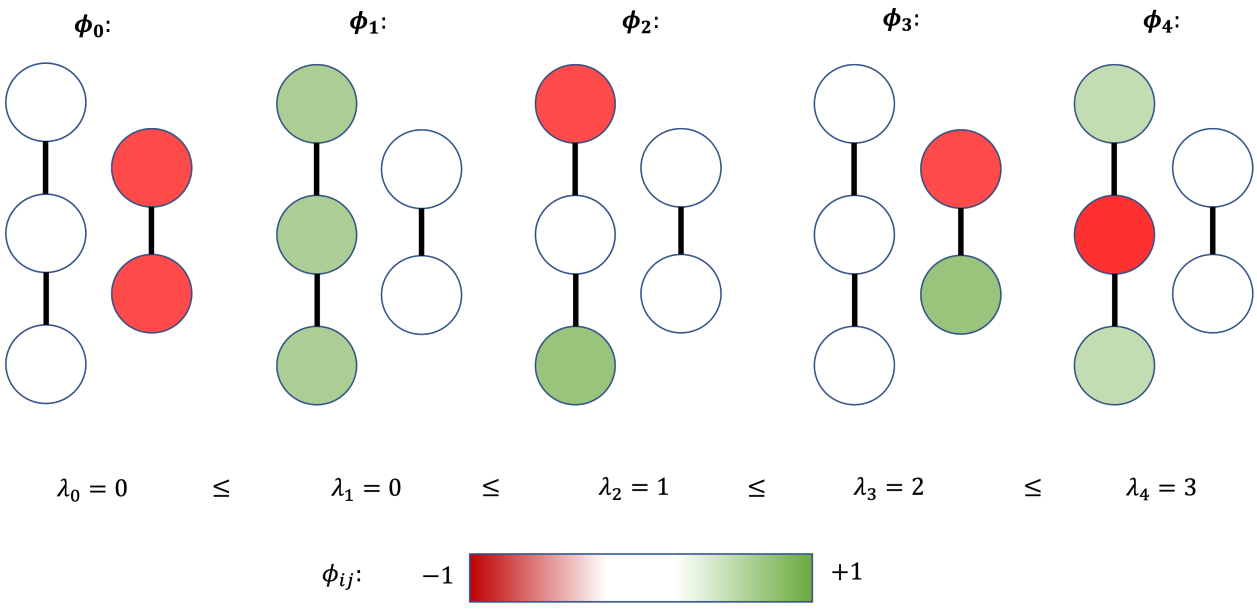


Figure 2.3: Real visualization of the Laplacian eigenvectors in non-decreasing order of eigenvalue. The lower the eigenvalue, the smoother the values transition from node to node.

operator, we interchangeably use the notion of Laplacian eigenvectors or eigenfunctions ϕ in this work. See Figure 2.3 for visualizing the eigenvectors and how the eigenvalue impacts the smoothness of the changing values.

These facts about eigenvectors allow us to more easily interpret them as signals propagating across the graph with frequency proportional to their eigenvalue (which is why we interchange the two terms). These signals can be used as stand-alone properties of the graph and are highly useful for graph clustering tasks as well as defining positional encodings.

2.2 Applications of Laplacian Eigenvectors

Here, we present three simple, yet highly effective, applications of the Laplace Eigenvectors for graph-related tasks.

2.2.1 Counting the number of connected components

One aspect of the Laplace decomposition left out of the previous section is that it is possible to observe geometric multiplicities greater than 1 for the various eigenvalues of a graph. In other words, different eigenvectors of the same Laplacian can have the exact same eigenvalues.

Surprisingly, the geometric multiplicity of the trivial eigenvalue λ_0 can be used to count the number of disconnected components in a graph [23]. This can be simple to understand when we realize that the trivial eigenvector only needs to assign the same constant value to neighboring nodes to achieve the global minimum in 2.4:

$$\boldsymbol{\phi}_{\mathbf{0}}^T \boldsymbol{L} \boldsymbol{\phi}_{\mathbf{0}} = \sum_{(i,j) \in E} (\boldsymbol{\phi}_{\mathbf{0}}[i] - \boldsymbol{\phi}_{\mathbf{0}}[j])^2 = 0$$

If the graph only has only a single connected component (it is possible to reach every node in the graph from any other), the only eigenvector solution is the one that assigns a constant value to all nodes.

However, if there are multiple connected components in the graph, there will exist many more solutions to 2.4 which achieve the same value of 0. Specifically, each solution only needs to assign a constant value to nodes *within a connected component*, and thus there will be as many solutions as there are connected components. Therefore, the geometric multiplicity of the trivial eigenvalue λ_0 can directly reveal how many connected components are in the graph. This was apparent in the example used in Figure 2.3.

2.2.2 Finding the Minimum Cut

The problem of finding the minimum cut of a graph consists of finding a partition of the nodes of a graph into sets S and $S \backslash V$ such that the number of edges going across the partition is minimized.

Solving this problem is actually analogous to optimizing equation 2.4, and can be solved by using the second-smallest (or first non-zero) eigenvector of the Laplacian [50].

Assuming the graph has one connected component, we can use the first non-zero eigenvector ϕ_1 in the following manner to assign nodes to the proper set which solves the minimum cut problem:

$$\begin{cases} u \in S, & \text{if } \phi_1[u] \geq 0 \\ u \in S \backslash V, & \text{if } \phi_1[u] < 0 \end{cases} \tag{2.5}$$

2.2.3 Generalized Spectral Clustering

The previous problem can be generalized to the problem of clustering the nodes of a graph into two separate components. In fact, we can take this problem a step further and use the first K eigenvectors of the Laplacian (excluding the non-trivial ones) to cluster the nodes of a graph into K separate components [23, 41].

All we need to is, for each node, assign a feature vector from the corresponding row of the first K non-trivial eigenvector columns in the Laplacian eigenvector matrix **U** (revert to 2.2 for visualizing), and run a K-means clustering algorithm on them.

2.3 Graph Learning Tasks

While classic machine learning methods are suited to model datapoints which lie in a Euclidean space, here we concern ourselves with ones that lie in the graph space: we seek to model components of graphs, or entire graphs themselves. Here, we will present the four most popular tasks for learning on graph-structured data.

2.3.1 Node Classification

In node classification, the goal is to predict labels for all nodes in the graph. As per regular machine learning tasks, these labels can be associated with binary classification, regression or categorization tasks.

Typically, for each graph, we are provided a set of training nodes $V_{train} \subset V$ whose labels are available, and a held out set of testing nodes, whose labels we are trying to predict.

Though node classification appears to be a standard supervised learning task, there are key differences that have made researchers refer to it as a *semi-supervised* learning task [54]. Importantly, the structure of the entire graph, including the unlabeled test nodes and their connecting edges, is available during training. This differs substantially from standard supervised learning, where all elements of test points (not just the label) are unobserved during training.

A classic example of node classification is labelling the topic of academic articles in a large citation graph, where nodes represent academic papers and edges represent citations [6, 31, 36, 40]. Other examples include classifying the function of proteins in the interactome [22], identifying bots in large social networks or the type of atom in a molecular graph.

Notions to exploit when building successful models for node classification include *homophily*, which is the concept that neighboring nodes typically share common features [38], and *structural equivalence*, the idea that nodes with similar neighborhoods tend to have similar attributes [16].

2.3.2 Link Prediction

Link prediction has a similar ambition as node classification, but the objective is to retrieve edges stripped from a graph. Formally, we are given a graph with a set of nodes V and a subset of training edges $E_{train} \subset E$. The goal is to predict the missing edges E_{test} to complete the graph.

Classic examples of link prediction including inferring the friendship status of individuals on an online network, predicting the side effects of drugs [59] and making recommendations to users on online platforms [55].

2.3.3 Graph Classification and Regression

In the previous two tasks, we were concerned with modelling components of a graph; nodes and edges. In this task, we are concerned with learning labels associated with entire graphs. In other words, each graph is its own independent datapoint associated with some real or discrete-valued label and, given training examples, we wish to learn functions which map graphs to these labels.

For example, given graphs representing molecules, we may be concerned with identifying those which inhibit HIV virus replication [26] or their toxicity and solubility [21].

This problem is the most straightforward analogy to supervised learning, as each graph is independent and identically distributed, and no information about the test set is accessible during training.

2.3.4 Clustering and Community Detection

Whilst the previously mentioned tasks are the clearest analogs to supervised learning, clustering and community detection are the analogs of unsupervised learning in the graph learning space.

In this task, the objective is very simple: given exclusively a graph G and its adjacency matrix, unravel its community structure and cluster the nodes of graphs into separate components.

In fact, the Laplacian eigenvectors discussed in section 2.1.3 are very useful for accomplishing this task, as it was revealed in sections 2.2.2 and 2.2.3 that they can help cluster nodes into separate partitions, establishing relative positions of nodes in a graph.

2.4 Graph Neural Networks

In the following section, we will discuss some of the most common frameworks underpinning *graph neural networks (GNN)*, which are a class of models used to learn deep rep-

resentations of graph-structured data. The inductive biases made for these models are typically designed with the objective of capturing information about the graph structures while infusing it with node and edge-level feature information.

Some of the main challenges GNNs overcome are invariance to the arbitrary ordering of nodes in the adjacency matrix (*permutation invariance*) and handling of variable number of neighboring nodes.

2.4.1 Message-Passing Neural Networks

Message-Passing Neural Networks (MPNNs) are the epitome of GNNs. Initially, each node $u \in V$ is given its node feature and/or statistical information as its representation $\mathbf{h_u^0}$. Then, with each MPNN iteration, each nodes' representation is updated based on information gathered from its neighborhood $N(u)$. The generalized MPNN framework can be visualized in Figure 2.4 and defined as follows [23]:

$$\mathbf{h_u^{k+1}} = \text{UPDATE}(\mathbf{h_u^k},\ \text{AGGREGATE}^k(\mathbf{h_v^k}, \forall v \in N(u)) \tag{2.6}$$

$$\mathbf{h_u^{k+1}} = \text{UPDATE}(\mathbf{h_u^k}, \mathbf{m_{N(u)}^k}) \tag{2.7}$$

where the UPDATE and AGGREGATE functions are the functions which vary across MPNN models, and $\mathbf{m_{N(u)}}$ is the aggregated *message*. Since the AGGREGATE function takes as input a set of nodes, these models are naturally invariant to their ordering.

The intuition behind this method is that with each iteration, nodes' representations contain increasing amounts of information about their neighborhood. Precisely, after a single iteration, each node knows about information from nodes in its 1-hop neighborhood. After a second iteration, they know information from their 2-hop neighborhood. Finally, after K iterations, each node has information from its K-hop neighborhood. This allows information to propagate across the graph.

After running K iterations of the method, the final embeddings $\mathbf{h_u^K}$ for each node u can be directly fed to a multi-layer perceptron (MLP) to accomplish the desired node-

level task. If the task is at the graph level, a pooling layer such as *sum* or *mean* is applied over the final node embeddings to generate a graph-level embedding, which can be subsequently also fed to an MLP.

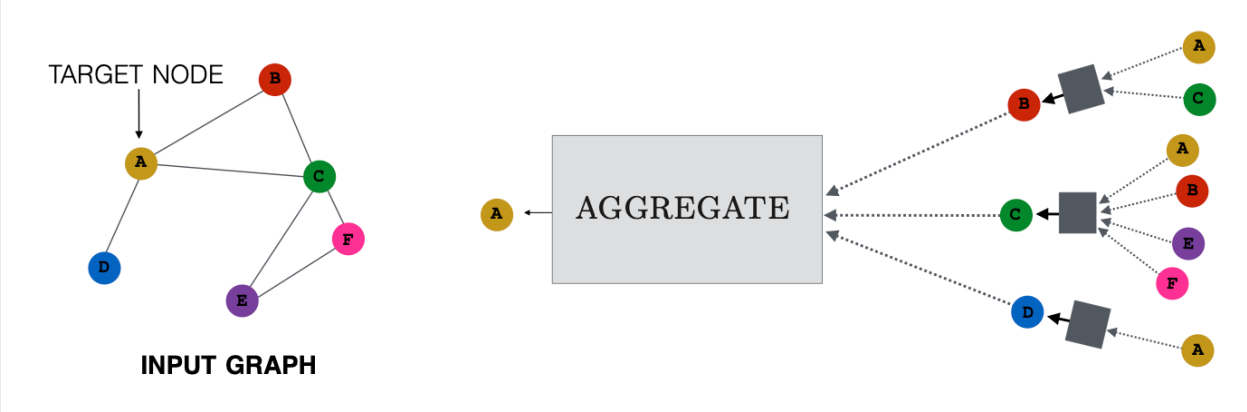


Figure 2.4: Intuition behind the message-passing mechanism. Node A aggregates information from nodes in its neighborhood who, in turn, also did the same. This allows information to be passed between nodes that may be far apart in the graph if multiple iterations are performed [23].

A concrete example of a basic MPNN can be defined as follows [23]:

$$\mathbf{h_u^{k+1}} = \sigma(\mathbf{W_{self}^k h_u^k} + \mathbf{W_{neigh}^k} \sum_{v \in N(u)} \mathbf{h_u^k}) \tag{2.8}$$

In this case, the AGGREGATE function to generate $\mathbf{m_{N(u)}}$ is to simply sum the neighbors

$$\mathbf{m_{N(u)}} = \sum_{N(u),} \mathbf{h_u}$$

while the UPDATE function is to simply learn linear projections of the current representation and the aggregated message before adding and passing them through a sigmoid:

$$\text{UPDATE}(\mathbf{h_u^k}, \mathbf{m_{N(u)}}) = \sigma(\mathbf{W_{self}^k h_u^k} + \mathbf{W_{neigh}^k m_{N(u)}})$$

.

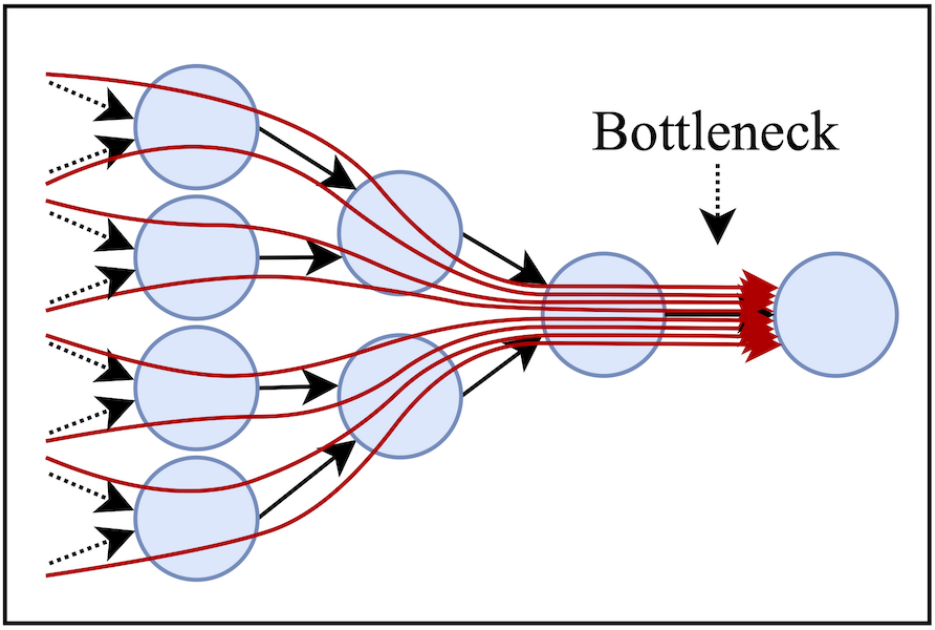


Figure 2.5: As multiple rounds of message-passing are performed, exponential amounts of information are being stored into constant-sized vector representations. This makes it challenging to propagate information between distant nodes [1].

Despite being popular, it is well recognized that message-passing has inherent limitations. For one, its expressive power appears inexorably bounded by the Weisfeiler-Lehman isomorphism hierarchy [37, 39, 53]. Furthermore, MPNNs are known to suffer from pathologies, such as *oversmoothing*, where repeated aggregations of local information tend to make representations increasingly indistinguishable, and *over-squashing*, where the exponential blow-up in computation paths as the model depth increases makes it challenging to capture long-range dependencies [1]. See Figure 2.5 for visualizing the latter.

2.4.2 Attention-based Graph Neural Networks

A defining feature of MPNN models is that the neighbors are treated equally during the aggregation layer. This is clear in the summation aggregator:

$$\mathbf{m_{N(u)}} = \sum_{N(u)} \mathbf{h_u}.$$

To improve the performance of MPNNs, a basic idea is to apply *attention* [2] to weigh the importance of each neighbor during the aggregation step. In fact, this was first employed in the Graph Attention Network (GAT) [49] with much success:

$$\mathbf{m_{N(u)}} = \sum_{N(u)} \alpha_{u,v} \mathbf{h_u},$$

where the attention weight $\alpha_{u,v}$ are defined as follows:

$$\alpha_{u,v} = \frac{\exp(\mathbf{a^T}[\mathbf{W h_u} || \mathbf{W h_v}])}{\sum_{v' \in N(u)} \exp(\mathbf{a^T}[\mathbf{W h_u} || \mathbf{W h'_v}])},$$

where $\mathbf{a}$ is a learnable vector, $\mathbf{W}$ is a learnable matrix and $||$ denotes concatenation.

This method is highly useful for increasing the expressivity of the GNN model and was shown to be superior to MPNN on several key tasks, but continues to suffer from the same limitations as MPNNs due to the nature of sparse aggregation.

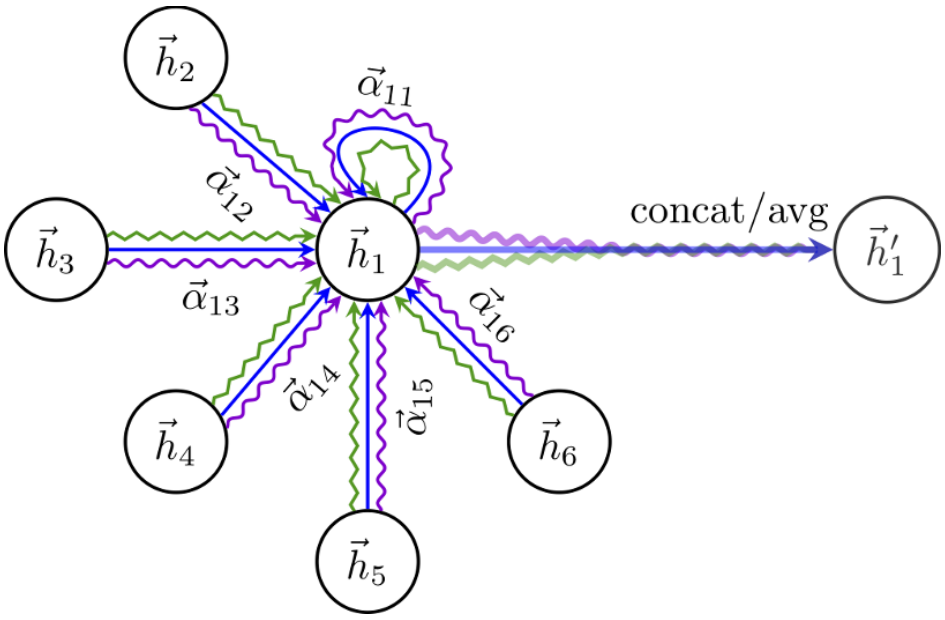


Figure 2.6: Visualizing the weighting of neighbors during aggregation in GAT [49].

2.4.3 Graph Transformers

Transformers [48] are a trending topic in the Deep Learning community, with its applications being explored beyond its intentions for NLP in areas including computer vision [17] and, recently, graphs [18, 56].

In particular, the Graph Transformer (GT) model [18] was proposed as a generalization of Transformers to graphs. The method incorporates the learnable K, Q, V matrix multiplications from the original Transformer architecture [48] to define attention and considers substituting the original sinusoids with Laplace eigenvectors as positional encodings.

Specifically, the authors propose selecting the first k non-trivial eigenvectors of the Laplacian, passing them through a linear transformation and adding them to the initial node representations to infuse a relative distance notion:

$$\boldsymbol{\phi}_{\boldsymbol{i}}^{\mathbf{0}} = \boldsymbol{C}^{\mathbf{0}}\boldsymbol{\phi}_{\boldsymbol{i}} + \mathbf{c}^{\mathbf{0}} \tag{2.9}$$

$$\mathbf{h}_{\mathbf{i}}^{\mathbf{0}} = \hat{\mathbf{h}}_{\mathbf{i}}^{\mathbf{0}} + \boldsymbol{\phi}_{\boldsymbol{i}}^{\mathbf{0}} \tag{2.10}$$

where $\mathbf{C^0} \in \mathbb{R}^{d \times k}$ and $\mathbf{c^0} \in \mathbb{R}^d$ are a learnable matrix and vector respectively, $\boldsymbol{\phi_i} \in \mathbb{R}^k$ are node i's first k eigenvector entries and $\hat{\boldsymbol{h}}_{\boldsymbol{i}} \in \mathbb{R}^d$ is node i's initial features. Using these Laplace eigenvectors as positional features is reminiscent of the Generalized Spectral Clustering algorithm detailed in Section 2.2.3, where it was revealed that they are highly useful for learning graph structural information on their own. Details on the graph attention mechanism are described in Section 4.3.

In their work, the authors also experimented with the intended usage of Transformers by fully connecting the graph to overcome the pathologies of MPNNs, but observed very poor performance in doing so.

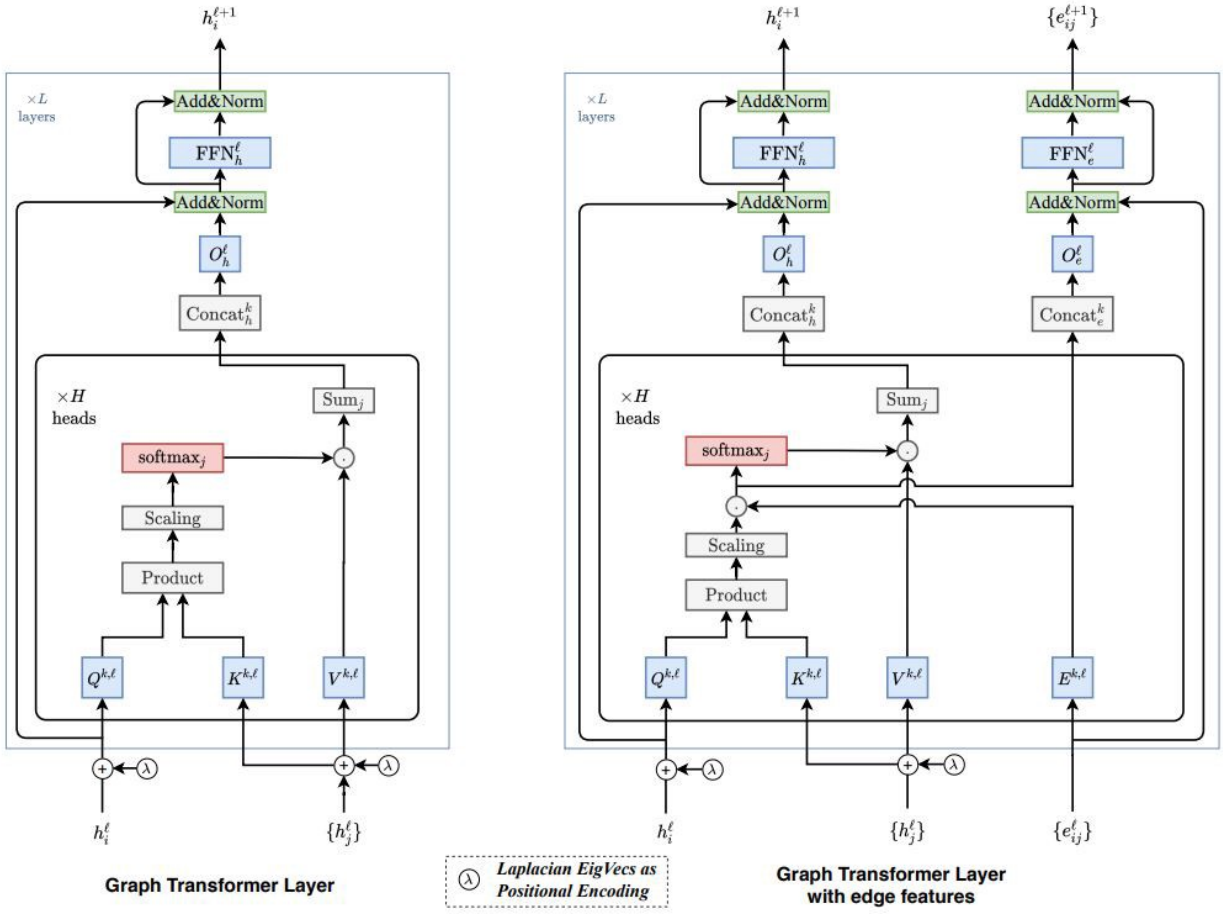


Figure 2.7: An overview of the GT model, which generalizes the attention mechanism used in Transformers to GNNs and employs a linear transformation of the first k non-trivial Laplace eigenvectors as substitutes for positional encodings.

Chapter 3

Theoretical Motivations

There can be a significant loss in structural information if naively generalizing Transformers to graphs. To preserve this information as well as local connectivity, previous studies [18, 49] have proposed to use the eigenfunctions of their Laplacian as positional encodings. Taking this idea further by using the full expressivity of eigenfunctions as positional encodings, we can propose a principled way of understanding graph structures using their spectra. The advantages of our methods compared to previous studies [18,49] are shown in Table 3.1.

Table 3.1: Comparison of the properties of different graph Transformer models.

MODELS	GAT	GT sparse	GT full	SAN
Preserves local structure in attention	✓	✓	✗	✓
Uses edge features	✗	✓	✗	✓
Connects non-neighbouring nodes	✗	✗	✓	✓
Uses eigenvector-based PE for attention	✗	✓	✓	✓
Use a PE with structural information	✗	✓	✗	✓
Considers the ordering of the eigenvalues	✗	✓	✓	✓
Invariant to the norm of the eigenvector	-	✓	✓	✓
Considers the spectrum of eigenvalues	✗	✗	✗	✓
Considers variable # of eigenvectors	-	✗	✗	✓
Aware of eigenvalue multiplicities	-	✗	✗	✓
Invariant to the sign of the eigenvectors	-	✗	✗	✗

3.1 Absolute and relative positional encodings with eigenfunctions

The notion of positional encodings (PEs) in graphs is not a trivial concept, as there exists no canonical way of ordering nodes or defining axes. In this section, we describe the differences between using absolute and relative PEs and investigate how eigenfunctions of the Laplacian can be used to define them. We also look into how they can measure physical interactions between nodes and enable ”hearing” of specific sub-structures - similar to how the sound of a drum can reveal its structure.

3.1.1 Absolute vs relative positional encodings

It is important to understand that the original Transformer architecture employs positional information in absolute terms; every element in the sequence of length n is given its own unique position using sinusoids - creating n PEs. However, it is also possible to consider relative positions, in which n^2 PEs are used to describe the *distances between each element in the sequence*.

Previous lines of work have investigated the benefit of employing relative positions to enhance Transformers, by adding learnable vectors between sequence elements [45] that can be directly included in the attention computation, which was shown to be useful in several other tasks [24, 51].

This work presents a similar formulation in 4.2, which attempts to learn positional encodings based on relative positions to tune the self-attention by employing Laplacian eigenfunctions for graph-structured inputs.

3.1.2 Eigenvectors equate to sine functions over graphs

In the Transformer architecture, a fundamental aspect is the use of sine and cosine functions as PEs for sequences [48]. However, sinusoids cannot be clearly defined for arbitrary

graphs, since there is no clear notion of position along an axis. Instead, their equivalent is given by the eigenvectors ϕ of the graph Laplacian $\boldsymbol{L}$. Indeed, in a Euclidean space, the Laplacian (or Laplace) operator corresponds to the divergence of the gradient and its eigenfunctions are sine/cosine functions, with the squared frequencies corresponding to the eigenvalues (we sometimes interchange the two notions from here on). Hence, in the graph domain, the eigenvectors of the graph Laplacian are the natural equivalent of sine functions, and this intuition was employed in multiple recent works which use the eigenvectors as PEs for GNNs [19], for directional flows [5] and for Transformers [18].

Being equivalent to sine functions, we naturally find that the Fourier Transform of a function $\mathcal{F}[f]$ applied to a graph gives $\mathcal{F}[f](\lambda_i) = \langle f, \phi_i \rangle$, where the eigenvalue is considered as a position in the Fourier domain of that graph [8]. Thus, the eigenvectors are best viewed as vectors positioned on the axis of eigenvalues rather than components of a matrix as illustrated in Figure 3.1.

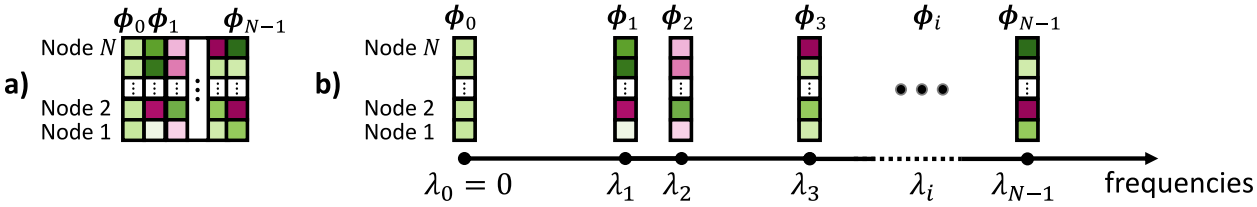


Figure 3.1: a) Standard view of the eigenvectors as a matrix. b) Eigenvectors ϕ_i viewed as vectors positionned on the axis of frequencies (eigenvalues).

3.1.3 What do eigenfunctions tell us about relative positions?

In addition to being the analog of sine functions, the eigenvectors of the Laplacian also hold important information about the physics of a system and can reveal distance metrics. This is not surprising as the Laplacian is a fundamental operator in physics and is notably used in Maxwell's equations [20] and the heat diffusion [8].

In electromagnetic theory, the (pseudo)inverse of the Laplacian, known in mathematics as the Green's function of the Laplacian [11], represents the electrostatic potential of a

given charge. In a graph, the same concept uses the pseudo-inverse of the Laplacian $\boldsymbol{G}$ and can be computed by its eigenfunctions. See equation 3.1 , where $\boldsymbol{G}(j_1, j_2)$ is the electric potential between nodes j_1 and j_2, $\hat{\phi}$ and $\hat{\lambda}_i$ are the i-th eigenvectors and eigenvalues of the symmetric Laplacian $\boldsymbol{D}^{\frac{-1}{2}}\boldsymbol{L}\boldsymbol{D}^{\frac{-1}{2}}$, and $\boldsymbol{D}$ is the degree matrix, and $\hat{\phi_{i,j}}$ the j-th row of the vector.

$$\boldsymbol{G}(j_1, j_2) = d_{j_1}^{\frac{1}{2}} d_{j_2}^{\frac{-1}{2}} \sum_{i>0} \frac{(\hat{\phi}_{i,j_1}\hat{\phi}_{i,j_2})^2}{\hat{\lambda}_i} \tag{3.1}$$

Further, the original solution of the heat equation given by Fourier relied on a sum of sines/cosines known as a Fourier series [9]. As eigenvectors of the Laplacian are the analogue of these functions in graphs, we find similar solutions. Knowing that heat kernels are correlated to random walks [5,8], we use the interaction between two heat kernels to define in equation 3.2 the diffusion distance d_D between nodes j_1, j_2 [8,12]. Similarly, the biharmonic distance d_B was proposed as a better measure of distances [35]. Here we use the eigenfunctions of the regular Laplacian $\boldsymbol{L}$.

$$d_D^2(j_1, j_2) = \sum_{k>0} e^{-2t\lambda_i}(\phi_{i,j_1} - \phi_{i,j_2})^2 \quad , \quad d_B^2(j_1, j_2) = \sum_{i>0} \frac{(\phi_{i,j_1} - \phi_{i,j_2})^2}{\lambda_i^2} \tag{3.2}$$

There are a few things to note from these equations. Firstly, they highlight the importance of pairing *eigenvectors and their corresponding eigenvalues* when supplying information about relative positions in a graph. Secondly, we notice that the product of eigenvectors is proportional to the electrostatic interaction, while the subtraction is proportional to the diffusion and biharmonic distances. Lastly, there is a consistent pattern across all 3 equations: smaller frequencies/eigenvalues are more heavily weighted when determining distances between nodes.

3.1.4 Hearing the shape of a graph and its sub-structures

Another well-known property of eigenvalues is how they can be used to discriminate between different graph structures and sub-structures, as they can be interpreted as the

frequencies of resonance of the graph. This led to the famous question about whether we can hear the shape of a drum from its eigenvalues [28], with the same questions also applying to geometric objects [14] and 3D molecules [44]. Various success was found with the eigenfunctions being used for partial functional correspondence [43], algorithmic understanding geometries [33], and style correspondence [14]. Examples of eigenvectors for molecular graphs are presented in Figure 3.2.

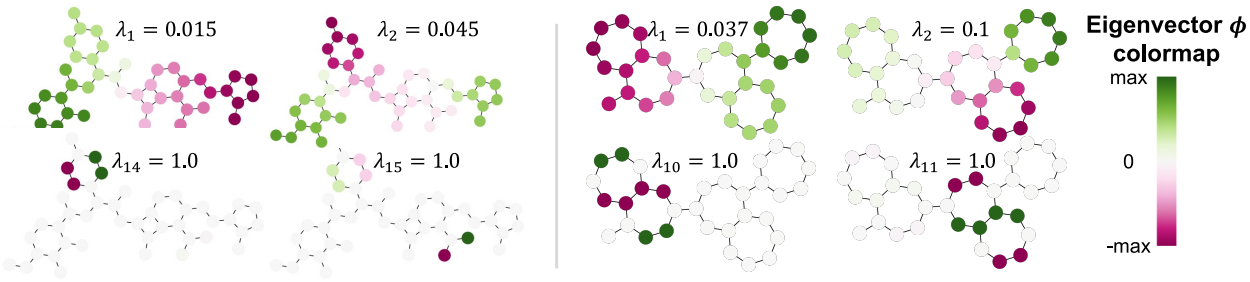


Figure 3.2: Examples of eigenvalues λ_i and eigenvectors ϕ_i for molecular graphs. The low-frequency eigenvectors ϕ_1, ϕ_2 are spread accross the graph, while higher frequencies, such as ϕ_{14}, ϕ_{15} for the left molecule or ϕ_{10}, ϕ_{11} for the right molecule, often resonate in local structures.

3.2 Laplace Eigenfunctions *etiquette*

In Euclidean space and sequences, using sinusoids as PEs is trivial: we can simply select a set of frequencies, compute the sinusoids, and add or concatenate them to the input embeddings, as is done in the original Transformer [48]. However, in arbitrary graphs, reproducing these steps is not as simple since each graph has a unique set of eigenfunctions. In the following section, we present key principles from spectral graph theory to consider when constructing PEs for graphs, most of which have been overlooked by prior methods. They include normalization, the importance of the eigenvalues and their multiplicities, the number of eigenvectors being variable, and sign ambiguities. Our LPE architectures, presented in section 4, aim to address them.

Normalization. Given an eigenvalue of the Laplacian, there is an associated eigenspace of dimension greater than 1. To make use of this information in our model, a single eigenvector has to be chosen. In our work, we use the L_2 normalization since it is compatible with the definition of the Green's function (3.1). Thus, we will always chose eigenvectors $\boldsymbol{\phi}$ such that $\langle \boldsymbol{\phi}, \boldsymbol{\phi} \rangle = 1$.

Eigenvalues. Another fundamental aspect is that the eigenvalue associated with each eigenvector supplies valuable information. An ordering of the eigenvectors based on their eigenvalue works in sequences since the frequencies are pre-determined. However, this assumption does not work in graphs since the eigenvalues in their spectrum can vary. For example, in Figure 3.2, we observe how an ordering would miss the fact that both molecules resonate at $\lambda = 1$ in different ways. Furthermore, it is possible to distinguish graphs solely on the spectrum of their eigenvalues.

Multiplicities. Another important problem with choosing eigenfunctions is the possibility of a high multiplicity of the eigenvalues, i.e. when an eigenvalue appears as a root of the characteristic polynomial more than once. In this case, the associated eigenspace may have dimension 2 or more as we can generate a valid eigenvector from any linear combination of eigenvectors with the same eigenvalue. This further complicates the problem of choosing eigenvectors for algorithmic computations and highlights the importance of having a model that can handle this ambiguity.

Variable number of eigenvectors. A graph G_i can have at most N_i linearly independent eigenvectors with N_i being its number of nodes. Most importantly, N_i can vary across all G_i in the dataset. Prior work [18] elected to select a fixed number k eigenvectors for each graph, where $k \leq N_i, \forall i$. This produces a major bottleneck when the smallest graphs have significantly fewer nodes than the largest graphs in the dataset since a very small proportion of eigenvectors will be used for large graphs. This inevitably causes loss of

information and motivates the need for a model which constructs fixed PEs of dimension k, where k does not depend on the number of eigenvectors in the graph.

Sign invariance. As noted earlier, there is a sign ambiguity with the eigenvectors. With the sign of ϕ being independent of its normalization, we are left with a total of 2^k possible combination of signs when choosing k eigenvectors of a graph. Previous work has proposed to do data augmentation by randomly flipping the sign of the eigenvectors [5, 18, 19], and although it can work when k is small, it becomes intractable for large k.

3.3 Learning with Eigenfunctions

Learning generalizable information from eigenfunctions is fundamental to their successful usage. Here we detail important points that support it is possible to do so if done correctly.

Similar graphs have similar spectra. Thus, we can expect the network to transfer patterns across graphs through the similarity of their spectra. In fact, spectral graph theory tells us that the lowest and largest non-zero eigenvalues are both linked to the geometry of the graph (algebraic connectivity and spectral radius).

Eigenspaces contain geometric information. Spectral graph theory has studied the geometric and physical properties of graphs from their Laplacian eigenfunctions in depth. Developing a method that can use the full spectrum of a graph makes it theoretically possible capture this information. It us thus important to capture differences between the full eigenspaces instead of minor differences between specific eigenvalues or eigenvectors from graph to graph.

Learned positions are relative within graphs. Eigenspaces are used to understand the relationship between nodes within graphs, not across them. Proposed models should therefore only compare the eigenfunctions of nodes within graphs.

Chapter 4

Model Architecture

In this section, we propose an elegant architecture that can use the eigenfunctions as PEs while addressing the concerns raised in section 3.2. Our *Spectral Attention Network* (SAN) model inputs eigenfunctions of a graph and projects them into a learned positional encoding (LPE) of fixed size. The LPE allows the network to use up to the entire Laplace spectrum of each graph, learn how the frequencies interact, and decide which are most important for the given task.

We propose a two-step learning process summarized in Figure 4.1. The first step, depicted by blocks (c-d-e) in the figure, applies a Transformer over the eigenfunctions of each node to generate an LPE matrix for each graph. The LPE is then concatenated to the node embeddings (blocks g-h), before being passed to the Graph Transformer (block i). If the task involves graph classification or regression, the final node embeddings are subsequently passed to a final pooling layer.

4.1 LPE Transformer Over Nodes

Using Laplace encodings as node features is ubiquitous in the literature concerning the topic. Here, we propose a method for learning node PEs motivated by the principles from section 3.2. The idea of our LPE is inspired by Figure 3.1, where the eigenvectors ϕ are

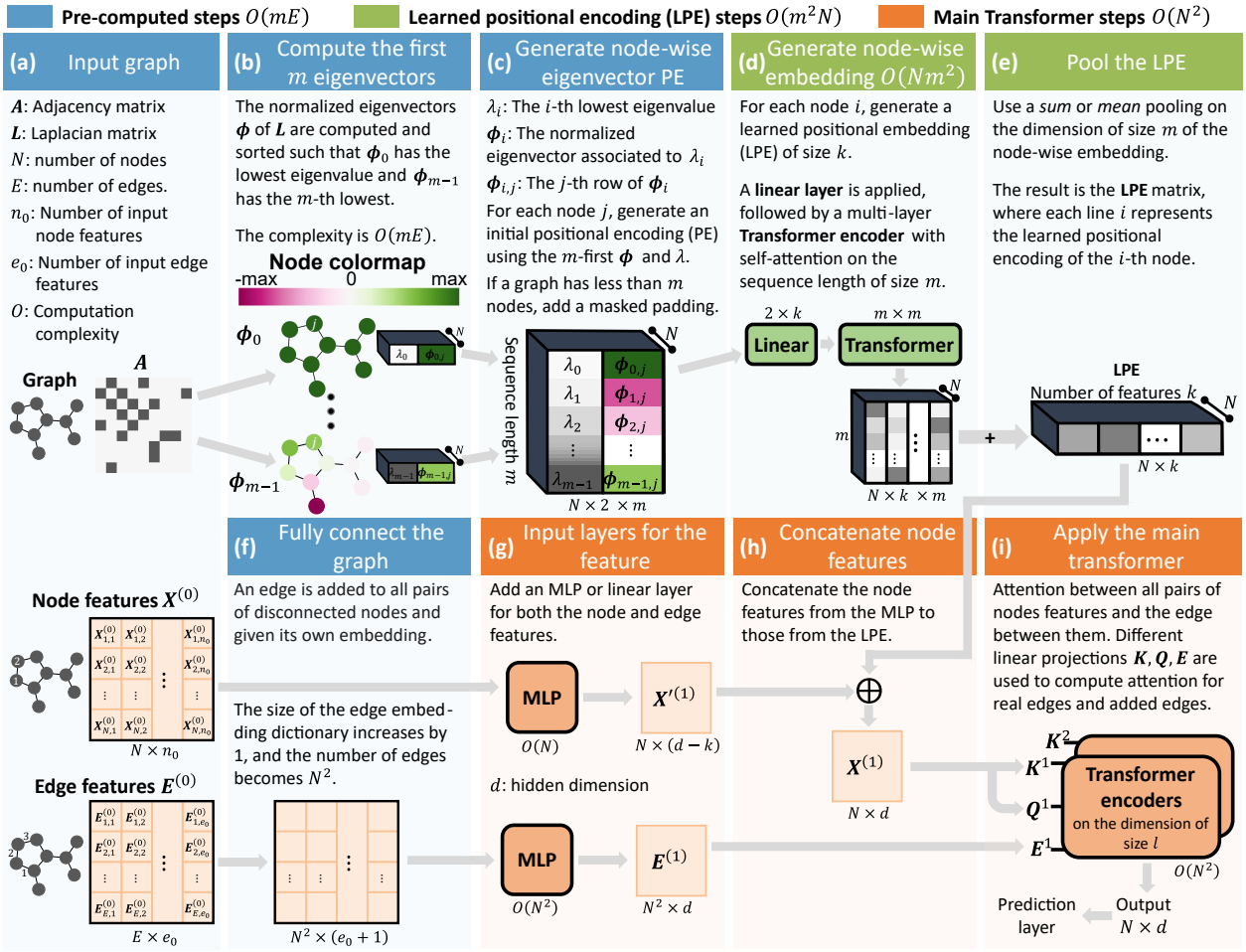


Figure 4.1: The proposed SAN model with the node LPE, a generalization of Transformers to graphs.

represented as a non-uniform sequence with the eigenvalue λ being the position on the frequency axis. With this representation, Transformers are a natural choice for processing them and generating a fixed-size PE.

The proposed LPE architecture is presented in Figure 4.2. First, we create an embedding matrix of size $2 \times m$ for each node j by concatenating the m-lowest eigenvalues with their associated eigenvectors. Here, m is a hyper-parameter for the maximum number of eigenvectors to compute and is analog to the variable-length sequence for a standard Transformer. For graphs where $m > N$, a masked-padding is simply added. Note that to capture the entire spectrum of all graphs, one can simply select m such that it is equal to the maximum number of nodes a graph has in the dataset. A linear layer is then applied

on the dimension of size 2 to generate new embeddings of size k. A Transformer Encoder then computes self-attention on the sequence of length m and hidden dimension k. Finally, a sum pooling reduces the sequence into a fixed k-dimensional node embedding.

The LPE model addresses key limitations of previous graph Transformers and is aligned with the first four *etiquettes* presented in section 3.2. By concatenating the eigenvalues with the normalized eigenvector, this model directly addresses the first three *etiquettes*. Namely, it **normalizes** the eigenvectors, pairs eigenvectors with their **eigenvalues** and treats **the number of eigenvectors as a variable**. Furthermore, the model is aware of **multiplicities** and has the potential to linearly combine or ignore some of the repeated eigenvalues.

However, this method still does not address the limitation that the sign of the pre-computed eigenvectors is arbitrary. To combat this issue, we randomly flip the sign of the pre-computed eigenvectors during training as employed by previous work [18,19], to promote invariance to the sign ambiguity.

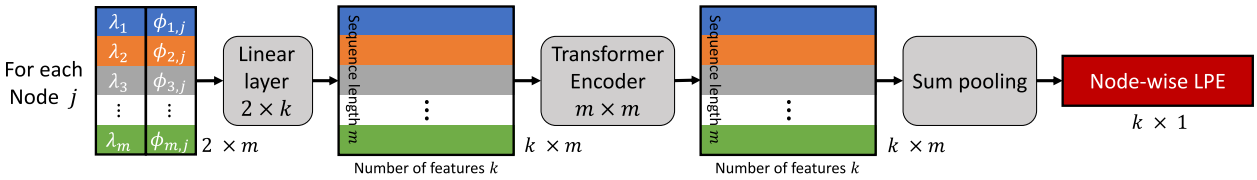


Figure 4.2: Learned positional encoding (LPE) architectures, with the model being aware of the graph's Laplace spectrum by considering m eigenvalues and eigenvectors, where we permit $m \leq N$, with N denoting the number of nodes. Since the Transformer loops over the nodes, each node can be viewed as an element of a batch to parallelize the computation. Here $\phi_{i,j}$ is the j-th element of the eigenvector paired to the i-th lowest eigenvalue λ_i.

4.2 LPE Transformer Over Edges

Here we present an alternative formulation for Laplace encodings. This method addresses the same issues as the LPE over nodes, but also resolves the eigenvector sign

ambiguity. Instead of encoding *absolute* positions as node features, the idea is to consider *relative* positions encoded as edge features.

Inspired by the physical interactions introduced in 3.1 and 3.2, we can take a pair of nodes (j_1, j_2) and obtain **sign-invariant** operators using the absolute subtraction $|\boldsymbol{\phi}_{i,j_1} - \boldsymbol{\phi}_{i,j_2}|$ and the product $\boldsymbol{\phi}_{i,j_1}\boldsymbol{\phi}_{i,j_2}$. These operators acknowledge that the sign of $\boldsymbol{\phi}_{i,j_1}$ at a given node j_1 is not important, but that the relative sign between nodes j_1 and j_2 is important. One might argue that we could directly compute the deterministic values from equations (3.1, 3.2) as edge features instead. However, our goal is to construct models that can learn which frequencies to emphasize and are not biased towards the lower frequencies — despite lower frequencies being useful in many tasks.

This approach is only presented thoroughly in appendix A, since it suffers from a major computational bottleneck compared to the LPE over nodes. In fact, for a fully-connected graph, there are N times more edges than nodes, thus the computation complexity is $O(m^2N^2)$, or $O(N^4)$ considering all eigenfunctions. The same limitation also affects memory and prevents the use of large batch sizes.

4.3 Main Graph Transformer

Our attention mechanism in the main Transformer is based on previous work [18], which attempts to repurpose the original Transformer to graphs by considering the graph structure and improving attention estimates with edge feature embeddings.

In the following, note that $\boldsymbol{h}_i^l$ is the i-th node's features at the l-th layer, and $\boldsymbol{e}_{ij}$ is the edge feature embedding between nodes i and j. Our model employs multi-head attention over all nodes:

$$\hat{\boldsymbol{h}}_i^{l+1} = \boldsymbol{O}_h^l \Big\Vert_{k=1}^{H} \Big(\sum_{j \in V} w_{ij}^{k,l} \boldsymbol{V}^{k,l} \boldsymbol{h}_j^l\Big) \tag{4.1}$$

where $\boldsymbol{O}_h^l \in \mathbb{R}^{d\times d}$, $\boldsymbol{V}^{k,l} \in \mathbb{R}^{d_k\times d}$, H denotes the number of heads, L the number of layers, and || concatenation. Note that d is the hidden dimension, while d_k is the dimension of a head ($\frac{d}{H} = d_k$).

A key addition from our work is the design of an architecture that performs full-graph attention while preserving local connectivity with edge features via two sets of attention mechanisms: one for nodes connected by real edges in the sparse graph and one for nodes connected by added edges in the fully-connected graph. The attention weights $w_{ij}^{k,l}$ in equation 4.1 at layer l and head k are given by:

$$\hat{\boldsymbol{w}}_{ij}^{k,l} = \begin{cases} \frac{\boldsymbol{Q}^{1,k,l}\boldsymbol{h}_i^l \circ \boldsymbol{K}^{1,k,l}\boldsymbol{h}_j^l \circ \boldsymbol{E}^{1,k,l}\boldsymbol{e}_{ij}}{\sqrt{d_k}} & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\boldsymbol{Q}^{2,k,l}\boldsymbol{h}_i^l \circ \boldsymbol{K}^{2,k,l}\boldsymbol{h}_j^l \circ \boldsymbol{E}^{2,k,l}\boldsymbol{e}_{ij}}{\sqrt{d_k}} & \text{otherwise} \end{cases} \tag{4.2}$$

$$w_{ij}^{k,l} = \begin{cases} \frac{1}{1+\gamma} \cdot \text{softmax}(\sum_{d_k} \hat{\boldsymbol{w}}_{ij}^{k,l}) & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\gamma}{1+\gamma} \cdot \text{softmax}(\sum_{d_k} \hat{\boldsymbol{w}}_{ij}^{k,l}) & \text{otherwise} \end{cases} \tag{4.3}$$

where $\circ$ denotes element-wise multiplication and $\boldsymbol{Q}^{1,k,l}$, $\boldsymbol{Q}^{2,k,l}$, $\boldsymbol{K}^{1,k,l}$, $\boldsymbol{K}^{2,k,l}$, $\boldsymbol{E}^{1,k,l}$, $\boldsymbol{E}^{2,k,l} \in \mathbb{R}^{d_k\times d}$. $\gamma \in \mathbb{R}^+$ is a hyperparameter which tunes the amount of bias towards full-graph attention, allowing flexibility of the model to different datasets and tasks where the necessity to capture long-range dependencies may vary. Note that, for numerical stability, the outputs of exponentiating the input to the softmax are clamped to values between -5 and $+5$ and that the keys, queries and edge projections are different for pairs of connected nodes ($\boldsymbol{Q}^1, \boldsymbol{K}^1, \boldsymbol{E}^1$) and disconnected nodes ($\boldsymbol{Q}^2, \boldsymbol{K}^2, \boldsymbol{E}^2$).

A multi-layer perceptron (MLP) with residual connections and normalization layers are then applied to update representations, in the same fashion as the GT method [18].

$$\hat{\hat{\boldsymbol{h}}}^{l+1} = \text{Norm}(\boldsymbol{h}_i^l + \hat{\boldsymbol{h}}_i^{l+1}), \quad \hat{\hat{\boldsymbol{h}}}_i^{l+1} = \boldsymbol{W}_2^l \text{ReLU}(\boldsymbol{W}_1^l \hat{\hat{\boldsymbol{h}}}_i^{l+1}), \quad \boldsymbol{h}_i^{l+1} = \text{Norm}(\hat{\hat{\boldsymbol{h}}}^{l+1} + \hat{\hat{\boldsymbol{h}}}_i^{l+1}) \tag{4.4}$$

with the weight matrices $\boldsymbol{W}_1^l \in \mathbb{R}^{2d \times d}$, $\boldsymbol{W}_2^l \in \mathbb{R}^{d \times 2d}$. Edge representations are not updated as it adds complexity with little to no performance gain. Bias terms are omitted for presentation.

4.4 Limitations

The first limitation of the node-wise LPE, and noted in Table 3.1 is the lack of sign invariance of the model. A random sign-flip of an eigenvector can produce different outputs for the LPE, meaning that the model needs to learn a representation invariant to these flips. We resolve this issue with the edge-wise LPE proposed in 4.2, but it comes at a computational cost.

Another limitation of the approach is the computational complexity of the LPE being $O(m^2N)$, or $O(N^3)$ if considering all eigenfunctions. Further, as nodes are batched in the LPE, the total memory on the GPU will be *num_params* * *num_nodes_in_batch* instead of *num_params* * *batch_size*. Although this is limiting, the LPE is not parameter hungry, with k usually kept around 16. Most of the model's parameters are in the *Main Graph Transformer* of complexity $O(N^2)$.

Despite Transformers having increased complexity, they managed to revolutionalize the NLP community. We argue that to shift away from the message-passing paradigm and generalize Transformers to graphs, it is natural to expect higher computational complexities. This is exacerbated by sequences being much simpler to understand than graphs due to their linear structure. Future work could overcome this by using variations of Transformers that scale linearly or logarithmically [46].

4.4.1 Theoretical properties of the architecture

Due to the full connectivity, it is trivial that our model does not suffer from the same limitations in expressivity as its convolutional/message-passing counterpart.

WL test and universality. The DGN paper [5] showed that using the eigenvector ϕ_1 is enough to distinguish some non-isomorphic graphs indistinguishable by the 1-WL test.

Given that our model uses the full set of eigenfunctions, and given enough parameters, our model can distinguish any pair of non-isomorphic graphs and is more powerful than any WL test in that regard. However, this does not solve the graph isomorphism problem in polynomial time; it only approximates a solution, and the number of parameters required is unknown and possibly non-polynomial. In appendix C, we present a proof of our statement, and discuss why the WL test is not well suited to study the expressivity of graph Transformers due to their universality.

Reduced over-squashing. Over-squashing represents the difficulty of a graph neural network to pass information to distant neighbours due to the exponential blow-up in computational paths [1].

For the fully-connected network, it is trivial to see that over-squashing is non-existent since there are direct paths between distant nodes.

Physical interactions. Another point to consider is the ability of the network to learn physical interactions between nodes. This is especially important when the graph models physical, chemical, or biological structures, but can also help understanding pixel interaction in images [3, 4]. Here, we argue that our SAN model, which uses the Laplace spectrum more effectively, can learn to mimic the physical interactions presented in section 3.1.3. This contrasts with the convolutional approach that requires deep layers for the receptive field to capture long-distance interactions. It also contrasts with the GT model [18], which does not use eigenvalues or enough eigenfunctions to properly model physical interactions in early layers. However, due to the lack of sign-invariance in the proposed node-wise LPE, it is difficult to learn these interactions accurately. The edge-wise LPE (section 4.2) could be better suited for the problem, but it suffers from higher computational complexity.

Chapter 5

Experimental Results

The model is implemented in PyTorch [42] and DGL [52] and tested on established benchmarks from [19] and [26] provided under MIT license. Specifically, we applied our method on ZINC, PATTERN, CLUSTER, MolHIV and MolPCBA [1], while following their respective training protocols with minor changes, as detailed in the appendix B.1. The computation time and hardware is provided in appendix B.4.

We first conducted an ablation study to fairly compare the benefits of using full attention and/or the node LPE. We then took the best-performing model, tuned some of its hyperparameters, and matched it up against the current state-of-the-art methods. Since we use a similar attention mechanism, our code was developed on top of the code from the GT paper [18], provided under the MIT license.

5.1 Sparse vs. Full Attention

To study the effect of incorporating full attention, we present an ablation study of the γ parameter in Figure 5.1. We remind readers that γ is used in equation 4.3 to balance between *sparse* and *full* attention. Setting $\gamma = 0$ strictly enables sparse attention, while $\gamma = 1$ does not bias the model in any direction.

[1] MolPCBA is only included in the *Comparison to the state-of-the-art* section due to its computational burden.

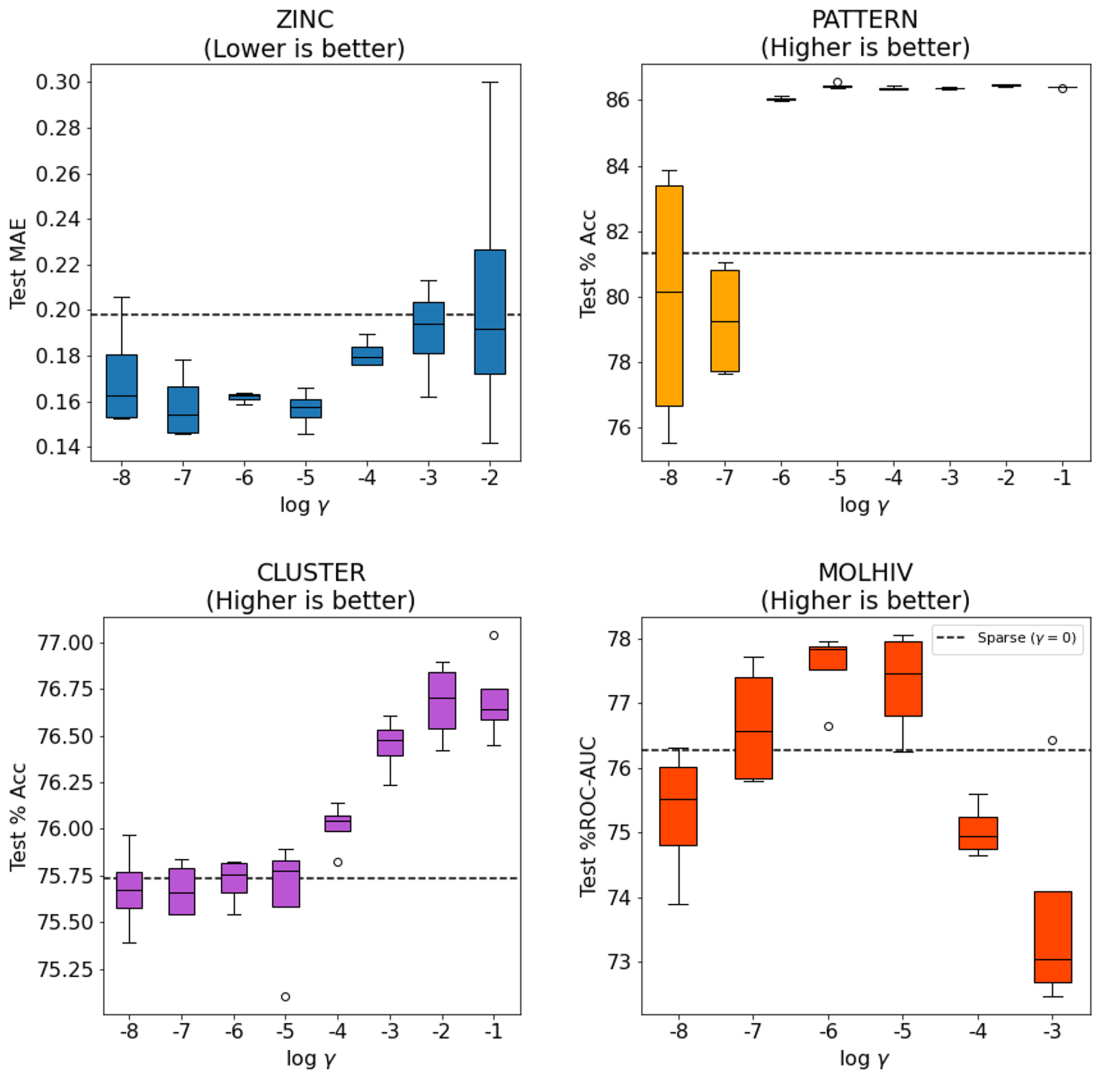


Figure 5.1: Effect of the γ parameter on the performance across datasets from [19, 26], using the Node LPE. Dotted black lines indicate sparse attention, which is equivalent to setting $\gamma = 0$. Each box plot consists of 4 runs, with different seeds (except MolHIV). Test results are presented at the best validation epoch for a given run. Note that this experiment serves solely to present the effect of tuning γ.

It is apparent that molecular datasets, namely ZINC and MOLHIV, benefit less from full attention, with the best parameter being $\log \gamma \in (-7, -5)$. On the other hand, the larger SBM datasets (PATTERN and CLUSTER) benefit from a higher γ value. This can be explained by the fact that molecular graphs rely more on understanding local structures such as the presence of rings and specific bonds, especially in the artificial task

Model details		ZINC	PATTERN	CLUSTER	MOLHIV
Connectivity	**LPE**	**MAE**	**% ACC**	**% ACC**	**% ROC-AUC**
Sparse	-	0.267 $\pm$ 0.032	83.613 $\pm$ 0.663	75.683 $\pm$ 0.098	73.46 $\pm$ 0.71
Sparse	Node	0.198 $\pm$ 0.004	81.329 $\pm$ 2.150	75.738 $\pm$ 0.106	76.61 $\pm$ 0.62
Full	-	0.392 $\pm$ 0.055	86.322 $\pm$ 0.049	76.447 $\pm$ 0.177	73.84 $\pm$ 1.80
Full	Node	**0.157 $\pm$ 0.066**	**86.441 $\pm$ 0.040**	**76.691 $\pm$ 0.247**	**77.57 $\pm$ 0.61**

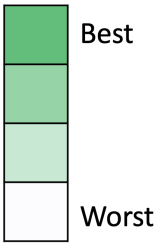


Figure 5.2: Ablation study on datasets from [19, 26] for the node LPE and full graph attention, with no hyperparameter tuning other than γ taken from the best validation scores in the experiment conducted for Figure 5.1. For a given dataset, all models use the same hyperparameters, but the hidden dimensions are adjusted to have $\sim 500k$ learnable parameters. Means and uncertainties are derived from four runs, with different seeds (except MolHIV).

from ZINC which relies on counting these specific patterns [19]. Furthermore, molecules are generally smaller than SBMs. As a result, we would expect less need for full attention, as information between distant nodes can be propagated with few iterations of even sparse attention. We also expect molecules to have fewer multiplicities, thus reducing the space of eigenvectors. Lastly, the performance gains in using full attention on the CLUSTER dataset can be attributed to it being a semi-supervised task, where some nodes within each graph are assigned their true labels. With full attention, every node receives information from the labeled nodes at each iteration, reinforcing confidence about the community they belong to.

In Figure 5.2, we present another ablation study to measure the impact of the node LPE in both the *sparse* and *full* architectures. We observe that the proposed node-wise LPE contributes significantly to the performance for molecular tasks (ZINC and MOLHIV), and believe that it can be attributed to the detection of substructures (see Figure 3.2). For PATTERN and CLUSTER, the improvement is modest as the tasks are simple clustering

[19]. Previous work even found that the optimal number of eigenvectors to construct PE for PATTERN is only 2 [18].

To compare the LPE to the simple concatenation of eigenvectors to node features, one can refer to results from the sparse GT model [18] presented in Figure 5.3.

5.1.1 Comparison to the state-of-the-art

When comparing to the state-of-the-art (SOTA) models in the literature in Figure 5.3, we observe that our SAN model consistently performs better on all synthetic datasets from [19], highlighting the strong expressive power of the model. On the MolHIV dataset, the performance on the test set is slightly lower than the SOTA. However, the model performs better on the validation set (85.30%) in comparison to PNA (84.25%) and DGN (84.70%). This can be attributed to a well-known issue with this dataset: the validation and test metrics have low correlation. In our experiments, we found higher test results with lower validation scores when restricting the number of epochs. Here, we also included results on the MolPCBA dataset, where we witnessed competitive results as well.

Other top-performing models, namely PNA [13] and DGN [5], use a message-passing approach [21] with multiple aggregators. When compared to attention-based models, SAN consistently outperforms the SOTA by a wide margin. To the best of our knowledge, SAN is the first fully-connected model to perform well on graph tasks, as is evident by the poor performance of the *GT (full)* model.

	ZINC	PATTERN	CLUSTER	MOLHIV	MOLPCBA
Model	MAE	% Acc	% Acc	% ROC-AUC	% AP
GCN	0.367 ± 0.011	71.892 ± 0.334	68.498 ± 0.976	76.06 ± 0.97	20.20 ± 0.24
GraphSage	0.398 ± 0.002	50.492 ± 0.001	63.844 ± 0.110	-	-
GatedGCN	0.282 ± 0.015	85.568 ± 0.088	73.840 ± 0.326	-	-
GatedGCN-PE	0.214 ± 0.013	86.508 ± 0.085	76.082 ± 0.196		
GIN	0.526 ± 0.013	85.387 ± 0.136	64.716 ± 1.553	75.58 ± 1.40	22.66 ± 0.28
PNA	0.142 ± 0.010	-	-	79.05 ± 1.32	28.38 ± 0.35
DGN	-	-	-	**79.70 ± 0.97**	**28.85 ± 0.30**
Attention-based					
GAT	0.384 ± 0.007	78.271 ± 0.186	70.587 ± 0.447	-	-
GT (sparse)	0.226 ± 0.014	84.808 ± 0.068	73.169 ± 0.662	-	-
GT (full)	0.598 ± 0.049	56.482 ± 3.549	27.121 ± 8.471	-	-
SAN	**0.139 ± 0.006**	**86.581 ± 0.037**	**76.691 ± 0.65**	77.85 ± 0.247	27.65 ± 0.42

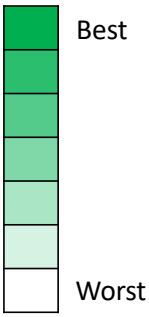


Figure 5.3: Comparing our tuned model on datasets from [19, 26], against GCN [30], GraphSage [22], GIN [53], GAT [49], GatedGCN [7], PNA [13], and DGN [5]. Means and uncertainties are derived from four runs with different seeds, except MolHIV which uses 10 runs with identical seed. The number of parameters is fixed to $\sim 500k$ for ZINC, PATTERN and CLUSTER.

Chapter 6

Conclusion

In summary, we presented the SAN model for graph neural networks, a new Transformer-based architecture that is aware of the Laplace spectrum of a given graph from the learned positional encodings. The model was shown to perform on par or better than the SOTA on multiple benchmarks and outperforms other Attention-based models by a large margin. As is often the case with Transformers, the current model suffers from a computational bottleneck, and we leave it for future work to implement variations of Transformers that scale linearly or logarithmically. This will enable the edge-wise LPE presented in appendix A, a theoretically more powerful version of the SAN model. It would also be worth exploring the employability of a learnable γ parameter.

Societal Impact. The presented work is focused on theoretical and methodological improvements to graph neural networks, so there are limited direct societal impacts. However, indirect negative impacts could be caused by malicious applications developed using the algorithm. One such example is the tracking of people on social media by representing their interaction as graphs, thus predicting and influencing their behavior towards an external goal. It also has an environmental impact due to the greater energy use that arises from the computational cost $O(m^2N + N^2)$ being larger than standard message passing or convolutional approaches of $O(E)$.

Bibliography

[1] ALON, U., AND YAHAV, E. On the bottleneck of graph neural networks and its practical implications. *arXiv:2006.05205 [cs, stat]* (2020).

[2] BAHDANAU, D., CHO, K., AND BENGIO, Y. Neural machine translation by jointly learning to align and translate, 2016.

[3] BEAINI, D., ACHICHE, S., DUPERRÉ, A., AND RAISON, M. Deep green function convolution for improving saliency in convolutional neural networks. *The Visual Computer 37*, 2 (2020), 227–244.

[4] BEAINI, D., ACHICHE, S., AND RAISON, M. Improving convolutional neural networks via conservative field regularisation and integration.

[5] BEAINI, D., PASSARO, S., LÉTOURNEAU, V., HAMILTON, W. L., CORSO, G., AND LIÒ, P. Directional graph networks. *ICML2021* (2021).

[6] BHATTACHARYA, I., AND GETOOR, L. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data 1*, 1 (mar 2007), 5–es.

[7] BRESSON, X., AND LAURENT, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553* (2017).

[8] BRONSTEIN, M. M., BRUNA, J., LECUN, Y., SZLAM, A., AND VANDERGHEYNST, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine 34*, 4 (2017), 18–42.

[9] CAJORI, F. *A History of Mathematics*. AMS Chelsea Publishing Series. AMS Chelsea, 1999.

[10] CHANG, H., RONG, Y., XU, T., HUANG, W., SOJOUDI, S., HUANG, J., AND ZHU, W. Spectral graph attention network. *CoRR abs/2003.07450* (2020).

[11] CHUNG, F., AND YAU, S. T. Discrete green's functions. *Journal of Combinatorial Theory, Series A 91*, 1 (2000), 191–214.

[12] COIFMAN, R. R., AND LAFON, S. Diffusion maps. *Applied and Computational Harmonic Analysis 21*, 1 (2006), 5–30. Special Issue: Diffusion Maps and Wavelets.

[13] CORSO, G., CAVALLERI, L., BEAINI, D., LIÒ, P., AND VELIČKOVIĆ, P. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718* (2020).

[14] COSMO, L., PANINE, M., RAMPINI, A., OVSJANIKOV, M., BRONSTEIN, M. M., AND RODOLA, E. Isospectralization, or how to hear shape, style, and correspondence. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[15] DONG, Y., DING, K., JALAIAN, B., JI, S., AND LI, J. Graph neural networks with adaptive frequency response filter. *CoRR abs/2104.12840* (2021).

[16] DONNAT, C., ZITNIK, M., HALLAC, D., AND LESKOVEC, J. Spectral graph wavelets for structural role similarity in networks. *CoRR abs/1710.10321* (2017).

[17] DOSOVITSKIY, A., BEYER, L., KOLESNIKOV, A., WEISSENBORN, D., ZHAI, X., UNTERTHINER, T., DEHGHANI, M., MINDERER, M., HEIGOLD, G., GELLY, S., USZKOREIT, J., AND HOULSBY, N. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[18] DWIVEDI, V. P., AND BRESSON, X. A generalization of transformer networks to graphs, 2020.

[19] DWIVEDI, V. P., JOSHI, C. K., LAURENT, T., BENGIO, Y., AND BRESSON, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982* (2020).

[20] FEYNMAN, R. P., LEIGHTON, R. B., AND SANDS, M. *The Feynman lectures on physics; New millennium ed.* Basic Books, New York, NY, 2010. Originally published 1963-1965.

[21] GILMER, J., SCHOENHOLZ, S. S., RILEY, P. F., VINYALS, O., AND DAHL, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 1263–1272.

[22] HAMILTON, W., YING, Z., AND LESKOVEC, J. Inductive representation learning on large graphs. In *Advances in neural information processing systems* (2017), pp. 1024–1034.

[23] HAMILTON, W. L. *Graph Representation Learning*. Morgan and Claypool, 2020.

[24] HELLENDOORN, V. J., SUTTON, C., SINGH, R., MANIATIS, P., AND BIEBER, D. Global relational models of source code. In *International Conference on Learning Representations* (2020).

[25] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[26] HU, W., FEY, M., ZITNIK, M., DONG, Y., REN, H., LIU, B., CATASTA, M., AND LESKOVEC, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687* (2020).

[27] JIN, W., BARZILAY, R., AND JAAKKOLA, T. Junction tree variational autoencoder for molecular graph generation. *arXiv:1802.04364 [cs, stat]* (2018).

[28] KAC, M. Can one hear the shape of a drum? *The American Mathematical Monthly 73*, 4 (1966), 1.

[29] KAZI, A., COSMO, L., NAVAB, N., AND BRONSTEIN, M. Differentiable graph module (dgm) graph convolutional networks. *arXiv preprint arXiv:2002.04999* (2020).

[30] KIPF, T. N., AND WELLING, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[31] KIPF, T. N., AND WELLING, M. Semi-supervised classification with graph convolutional networks. *CoRR abs/1609.02907* (2016).

[32] KREUZER, D., BEAINI, D., HAMILTON, W. L., LÉTOURNEAU, V., AND TOSSOU, P. Rethinking graph transformers with spectral attention. *Advances in nueral information processing systems (2021) abs/2106.03893* (2021).

[33] LEVY, B. Laplace-beltrami eigenfunctions towards an algorithm that "understands" geometry. In *IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)* (2006), pp. 13–13.

[34] LI, P., WANG, Y., WANG, H., AND LESKOVEC, J. Distance encoding: Design provably more powerful neural networks for graph representation learning, 2020.

[35] LIPMAN, Y., RUSTAMOV, R. M., AND FUNKHOUSER, T. A. Biharmonic distance. *ACM Trans. Graph. 29*, 3 (July 2010).

[36] LU, Q., AND GETOOR, L. Link-based classification.

[37] MARON, H., BEN-HAMU, H., SERVIANSKY, H., AND LIPMAN, Y. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136* (2019).

[38] MCPHERSON, M., SMITH-LOVIN, L., AND COOK, J. Birds of a feather: Homophily in social networks. *Annual Review of Sociology 27* (01 2001), 415–.

[39] MORRIS, C., RITZERT, M., FEY, M., HAMILTON, W. L., LENSSEN, J. E., RATTAN, G., AND GROHE, M. Weisfeiler and leman go neural: Higher-order graph neural

networks. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 4602–4609.

[40] NAMATA, G. M., LONDON, B., GETOOR, L., AND HUANG, B. Query-driven active surveying for collective classification. In *Workshop on Mining and Learning with Graphs* (2012).

[41] NG, A. Y., JORDAN, M. I., AND WEISS, Y. On spectral clustering: Analysis and an algorithm. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic* (Cambridge, MA, USA, 2001), NIPS'01, MIT Press, p. 849–856.

[42] PASZKE, A., GROSS, S., CHINTALA, S., CHANAN, G., YANG, E., DEVITO, Z., LIN, Z., DESMAISON, A., ANTIGA, L., AND LERER, A. Automatic differentiation in pytorch.

[43] RODOLÀ, E., COSMO, L., BRONSTEIN, M. M., TORSELLO, A., AND CREMERS, D. Partial functional correspondence, 2015.

[44] SCHRIER, J. Can one hear the shape of a molecule (from its coulomb matrix eigenvalues)? *Journal of Chemical Information and Modeling 60*, 8 (2020), 3804–3811. Publisher: American Chemical Society.

[45] SHAW, P., USZKOREIT, J., AND VASWANI, A. Self-attention with relative position representations. *CoRR abs/1803.02155* (2018).

[46] TAY, Y., DEHGHANI, M., BAHRI, D., AND METZLER, D. Efficient transformers: A survey, 2020.

[47] VAN DAM, E. R., AND HAEMERS, W. H. Which graphs are determined by their spectrum? *Linear Algebra and its Applications 373* (2003), 241–272.

[48] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L., AND POLOSUKHIN, I. Attention is all you need. *CoRR abs/1706.03762* (2017).

[49] VELIČKOVIĆ, P., CUCURULL, G., CASANOVA, A., ROMERO, A., LIO, P., AND BENGIO, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[50] VON LUXBURG, U. A tutorial on spectral clustering. *CoRR abs/0711.0189* (2007).

[51] WANG, B., SHIN, R., LIU, X., POLOZOV, O., AND RICHARDSON, M. RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. *CoRR abs/1911.04942* (2019).

[52] WANG, M., ZHENG, D., YE, Z., GAN, Q., LI, M., SONG, X., ZHOU, J., MA, C., YU, L., GAI, Y., XIAO, T., HE, T., KARYPIS, G., LI, J., AND ZHANG, Z. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

[53] XU, K., HU, W., LESKOVEC, J., AND JEGELKA, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[54] YANG, Z., COHEN, W. W., AND SALAKHUTDINOV, R. Revisiting semi-supervised learning with graph embeddings. *CoRR abs/1603.08861* (2016).

[55] YING, R., HE, R., CHEN, K., EKSOMBATCHAI, P., HAMILTON, W. L., AND LESKOVEC, J. Graph convolutional neural networks for web-scale recommender systems. *CoRR abs/1806.01973* (2018).

[56] YUN, C., BHOJANAPALLI, S., RAWAT, A. S., REDDI, S. J., AND KUMAR, S. Are transformers universal approximators of sequence-to-sequence functions?, 2020.

[57] YUN, C., CHANG, Y., BHOJANAPALLI, S., RAWAT, A. S., REDDI, S. J., AND KUMAR, S. $o(n)$ connections are expressive enough: Universal approximability of sparse transformers. *CoRR abs/2006.04862* (2020).

[58] YUN, S., JEONG, M., KIM, R., KANG, J., AND KIM, H. J. Graph transformer networks, 2020.

[59] ZITNIK, M., AGRAWAL, M., AND LESKOVEC, J. Modeling polypharmacy side effects with graph convolutional networks. *CoRR abs/1802.00543* (2018).

Appendices

Appendix A

LPE Transformer over Edges

Consider one of the most fundamental notions in physics; *Potential energy*. Interestingly, potential energy is always measured as a potential difference; it is not an inherent individual property, such as mass. Strikingly, it is also the *relative* Laplace embeddings of two nodes that paint the picture, as a node's Laplace embedding on its own reveals no information at all. With this in mind, we argue that Laplace positional encodings are more naturally represented as edge features, which encode a notion of *relative* position of the two endpoints in the graph. This can be viewed as a distance encoding, which was shown to improve the performance of node and link prediction in GNNs [34].

The formulation is very similar to the method for learning positional node embeddings. Here, a Transformer Encoder is applied on each graph by treating edges as a batch of variable size and eigenvectors as a variable sequence length. We again compute up to the m-lowest eigenvectors with their eigenvalues but, instead of directly using the eigenvector elements, we compute the following vectors:

$$|\phi_{:,j_1} - \phi_{:,j_2}| \qquad \text{(A.1)} \qquad\qquad \phi_{:,j_1} \circ \phi_{:,j_2} \qquad \text{(A.2)}$$

where ":" denotes along all up to m eigenvectors, and $\circ$ denotes element-wise multiplication. Note that these new vectors are completely invariant to sign permutations of the precomputed eigenvectors.

As per the LPE over nodes, the 3-length vectors are expanded with a linear layer to generate embeddings of size k before being input to the Transformer Encoder. The final embeddings are then passed to a sum pooling layer to generate fixed-size edge positional encodings, which are then used to compute attention weights in equation 4.2.

This method addresses **all etiquettes** raised in section 3.2. However, it suffers from a major computational bottleneck compared to the LPE over nodes. Indeed, for a fully-connected graph, there are N times more edges than nodes, thus the computation complexity is $O(m^2N^2)$, or $O(N^4)$ considering all eigenfunctions. This same limitation also affects the memory, as efficiently batching the N^2 edges will increase the memory consumption of the LPE by a drastic amount, preventing the model from using large batch sizes and making it difficult to train.

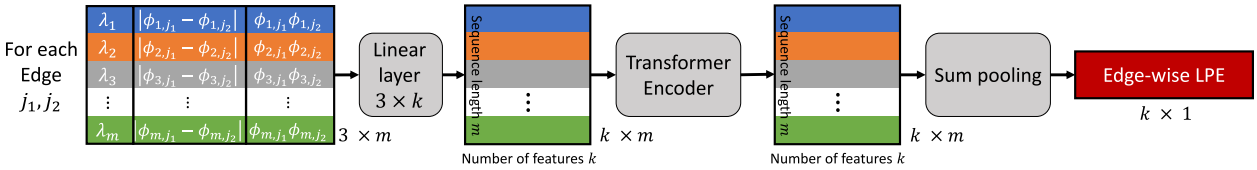


Figure A.1: Edge-wise Learned positional encoding (LPE) architectures, where the relative position is considered instead of the absolute position. The model is aware of the graph's Laplace spectrum by considering m eigenvalues and eigenvectors, where we permit $m \leq N$, with N denoting the number of nodes. Since the Transformer loops over the edges, each edge can be viewed as an element of a batch to parallelize the computation. The computational complexity is $O(m^2E)$ or $O(m^2N^2)$ for a fully-connected graph.

Appendix B

Implementation details

B.1 Benchmarks and datasets

To test our models' performance, we rely on standard benchmarks proposed by [19] and [26] and provided under the MIT license. In particular, we chose ZINC, PATTERN, CLUSTER, and MolHIV.

ZINC [19]. A synthetic molecular graph regression dataset, where the predicted score is given by the subtraction of computationally estimated properties $logP - SA$. Here, $logP$ is the computed octanol-water partition coefficient, and SA is the synthetic accessibility score [27].

CLUSTER [19]. A synthetic benchmark for node classification. The graphs are generated with Stochastic Block Models, a type of graph used to model communities in social networks. In total, 6 communities are generated and each community has a single node with its true label assigned. The task is to classify which nodes belong to the same community.

PATTERN [19]. A synthetic benchmark for node classification. The graphs are generated with Stochastic Block Models, a type of graph used to model communities in social net-

works. The task is to classify the nodes into 2 communities, testing the GNNs ability to recognize predetermined subgraphs.

MolHIV [26]. A real-world molecular graph classification benchmark. The task is to predict whether a molecule inhibits HIV replication or not. The molecules in the training, validation, and test sets are divided using a scaffold splitting procedure that splits the molecules based on their two-dimensional structural frameworks. The dataset is heavily imbalanced towards negative samples. It is also known that this dataset suffers from a strong de-correlation between validation and test set performance, meaning that more hyperparameter fine-tuning on the validation set often leads to lower test set results.

MolPCBA [26]. Another real-world molecular graph classification benchmark. The dataset is larger than MolHIV and applies a similar scaffold spliting procedure. It consists of multiple, extremely skewed (only 1.4% positivity) molecular classification tasks, and employs Average Precision (AP) over them as a metric.

B.2 Ablation studies

The results in Figures 5.1-5.2 are done as an ablation study with a minimal tuning of the hyperparameters of the network to measure the impact of the node LPE and full attention. A majority of the hyperparameters used were tuned in previous work [19]. However, we altered some of the existing parameters to accommodate the parameter-heavy LPE, and modified the Main Graph Transformer hidden dimension such that all models have approximately $\sim 500k$ parameters for a fair comparison. We present results for the full attention with the optimal γ value optimized on validation data for the Node LPE model. We did this to isolate the impact that the Node LPE has on improving full attention. Details concerning the model architecture parameters are visible in Figure B.1.

Attention	LPE	LPE layers	LPE dimension	GT layers	GT hidden dimension	#Parameters
ZINC						
†Sparse	-	-	-	6	96	511201
Sparse	Node	3	16	6	72	494865
Full	-	-	-	6	80	471361
Full	Node	3	16	6	64	508577
PATTERN						
Sparse	-	-	-	6	96	508634
Sparse	Node	3	16	6	72	493340
Full	-	-	-	6	80	469142
Full	Node	3	16	6	64	507202
CLUSTER						
Sparse	-	-	-	16	56	461348
Sparse	Node	1	16	16	56	530036
Full	-	-	-	16	48	450498
Full	Node	1	16	16	48	519186
MOLHIV						
Sparse	-	-	-	6	96	525985
Sparse	Node	2	16	6	80	503265
Full	-	-	-	6	80	483601
†Full	Node	2	16	6	72	528265

Figure B.1: Model architecture parameters for the ablation study. We modify the hidden dimensions of the Main Graph Transformer (GT) such that all models have $\sim 500k$ parameters for a fair comparison. Parameters were taken at the highest validation epoch performance.
†The batch size was doubled to ensure convergence of the model. All other parameters outside the GT hidden dimension are consistent within a dataset experiment.

For the training parameters, we employed an Adam optimizer with a learning rate decay strategy initialized in $\{10^{-3}, 10^{-4}\}$as per [19], with some minor modifications:

ZINC [19]. We selected an *initial learning rate* of 7×10^{-4} and increased the *patience* from 10 to 25 to ensure convergence.

PATTERN [19]. We selected an *initial learning rate* of 5×10^{-4}.

CLUSTER [19]. We selected an *initial learning rate* of 5×10^{-4} and reduced the *minimum learning rate* from 10^{-6} to 10^{-5} to speed up training time.

MolHIV [26]. We elected to use similar training procedures for consistency. We selected an *initial learning rate* of 10^{-4}, a *reduce factor* of 0.5, a *patience* of 20, a *minimum learning rate* of 10^{-5}, a *weight decay* of 0 and a *dropout* of 0.03.

B.3 SOTA Comparison study

For the results in Figure 5.3, we tuned some of the hyperparameters, using the following strategies. The optimal parameters are in **bold**.

ZINC. Due to the $500k$ parameter budget, we tuned the pairing $\{$*GT layers*, *GT hidden dimension*$\} \in \{\{6, 72\}, \{8, 64\},$ **{10,56}**$\}$ and *readout* $\in \{$"mean", **"sum"**$\}$

PATTERN. Due to the $500k$ parameter budget and long training times, we only tuned the pairing $\{$*GT layers*, *GT hidden dimension*$\} \in \{$**{4,80}**$, \{6, 64\}\}$

CLUSTER. Due to the $500k$ parameter budget and long training times, we only tuned the pairing $\{$*GT layers*, *GT hidden dimension*$\} \in \{\{12, 64\},$ **{16,48}**$\}$

MolHIV. With no parameter budget, we elected to do a more extensive parameter tuning in a two-step process while measuring validation metrics on 3 runs with identical seeds.

1. We tuned *LPE dimension* $\in \{8, \mathbf{16}\}$, *GT layers* $\in \{4, 6, 8, \mathbf{10}\}$, *GT hidden dimension* $\in \{48, \mathbf{64}, 72, 80, 96\}$
2. With the highest performing validation model from step 1, we then tuned *dropout* $\in \{0, \mathbf{0.01}, 0.025\}$ and *weight decay* $\in \{\mathbf{0}, 10^{-6}, 10^{-5}\}$

With the final optimized parameters, we reran 10 experiments with identical seeds.

MolPCBA. With no parameter budget, we elected to do a more extensive parameter tuning as well. We tuned *learning rate* $\in \{0.0001, \mathbf{0.0003}, 0.0005\}$, *dropout* $\in \{0, 0.1, 0.2, 0.3, 0.4, \mathbf{0.5}\}$, *GT layers* $\in \{2, 4, \mathbf{5}, 6, 8, 10, 12\}$, *GT layers* $\in \{128, 256, \mathbf{304}, 512\}$, *LPE layers* $\in \{8, \mathbf{10}, 12\}$ and *LPE dimension* $\in \{8, \mathbf{16}\}$

B.4 Computation details

Dataset	Resource	Cluster	GPU	Epoch/Total time
ZINC	Compute Canada	Graham	Tesla P100-PCIE (12 GB)	106s/17.88hrs
PATTERN	Compute Canada	Graham	Tesla P100-PCIE (12 GB)	340s/12.52hrs
CLUSTER	Compute Canada	Beluga	Tesla V100-SXM2 (16 GB)	433s/11.30hrs
MOLHIV	Compute Canada	Cedar	Tesla V100-SXM2 (32 GB)	204s/5.34hrs
MOLPCBA	Compute Canada	Cedar	Tesla V100-SXM2 (32 GB)	883s/48.02hrs

Figure B.2: Computational details for SOTA Comparison study.

Appendix C

Expressivity and complexity analysis of graph Transformers

In this section, we discuss how the universality of Transformers translates to graphs when using different node identifiers. Theoretically, this means that by simply labeling each node, Transformers can learn to distinguish any graph, and the WL test is no longer suited to study their expressivity.

Thus, we introduce the notion of learning complexity to better compare each architecture's ability to understand the space of isomorphic graphs. We apply the complexity analysis to the LPE and show that it can more easily capture the structure of graphs than a naive Transformer.

C.1 Universality of Transformers for sequence-to-sequence approximations

In recent work [56] [57], it was proven that Transformers are universal sequence-to-sequence approximators, meaning that they can encode any function that approximately maps any first sequence into a second sequence when given enough parameters. More formally, they proved the following theorems for the universality of Transformers:

Theorem 1 *For any $1 \leq p < \infty$, $\varepsilon > 0$ and any function $f : \boldsymbol{R}^{d \times n} \to \boldsymbol{R}^{d \times n}$ that is equivariant to permutations of the columns, there is a Transformer g such that the L^p distance between f and g is smaller than ε.*

Let $\boldsymbol{B}^n$ be the n-dimensional closed ball and denote by $C^0(\boldsymbol{B}^{d \times n}, \boldsymbol{R}^{d \times n})$ the set of all continuous functions of the ball to $\boldsymbol{R}^{d \times n}$. A Transformer with positional encoding g_p is a Transformer g such that to each input $\boldsymbol{X}$, a fixed learned positional encoding $\boldsymbol{E}$ is added such that $g_p(\boldsymbol{X}) = g(\boldsymbol{X} + \boldsymbol{E})$.

Theorem 2 *For any $1 \leq p < \infty$, $\varepsilon > 0$ and any function $f \in C^0(\boldsymbol{B}^{d \times n}, \boldsymbol{R}^{d \times n})$, there is a Transformer with positional encoding g such that the L^p distance between f and g is smaller than ε.*

C.2 Graph Transformers approximate solutions to the graph isomorphism problem

We now explore the consequences of the previous 2 theorems on the use of Transformers for graph representation learning. We first describe 2 types of Transformers on graphs; one for node and one for edge inputs. They will be used to deduce corollaries of theorems 1 and 2 for graph learning and later comparison with our proposed architecture. Assume now that all nodes of the graphs we consider are given an integer label in $\{1, ..., N\}$.

The naive edge transformer takes as input a graph represented as a sequence of ordered pairs $((i, j), \sigma_{i,j})$ with $i \leq j$ the indices of 2 vertices and $\sigma_{i,j}$ equal to 1 or 0 if the vertices i, j are connected or not. Recall there are $N(N-1)/2$ pairs of integers i, j in $\{1, ..., N\}$ with $i < j$ the indices of 2 vertices and $\sigma_{i,j}$ equal to 1 or 0 if the vertices i, j are connected or not. It is obvious that any ordering of these edge vectors describe the same graph. Recall there are $N(N-1)/2$ pairs of integers i, j in $\{1, ..., N\}$ with $i \leq j$. Consider the set of functions $f : \boldsymbol{R}^{N(N-1)/2 \times 2} \to \boldsymbol{R}^{N(N-1)/2 \times 2}$ that are equivariant to the permutations of

columns then theorem 1 says the function f can be approximated with arbitrary accuracy by Transformers on edge input.

The naive node Transformer can be defined as a Transformer with positional encodings. This graph Transformer will take as input the identity matrix and as positional encodings the padded adjacency matrix. This can be viewed as a one-hot encoding of each node's neighbors. Consider the set of continuous functions $f : \boldsymbol{R}^{N\times N} \to \boldsymbol{R}^{N\times N}$, then theorem 2 says the function f can be approximated with arbitrary accuracy by Transformers on node inputs.

From these two observations on the universality of graph Transformers, we get as a corollary that these 2 types of Transformers can approximate solutions of the graph isomorphism problem. In each case, pick a function that is invariant under node index permutations and maps non-isomorphic graphs to different values and apply theorem 1 or 2 that shows there is a Transformer approximating that function to an arbitrarily small error in the L^p distance. This is an interesting fact since it is known that the discrimination power of most message passing graph networks is upper bounded by the Weisfeiler-Lehman test which is unable to distinguish some graphs.

Here, we want to prove that given a unique node label, node connectivity, and the right architecture, Transformers can approximate a solution to the graph isomorphism problem.

Assume now that all nodes of the graphs we consider are given an integer label in $\{1, ..., N\}$. From theorem 1, we can deduce the following about Transformers on graphs. First, we consider the consequence of these theorems for classification of graph isomorphism classes using a Transformer on the edges. A graph will be represented as a sequence of ordered pairs $((i, j), \sigma_{i,j})$ with $i \leq j$ the indices of 2 vertices and $\sigma_{i,j}$ equal to 1 or 0 if the vertices i, j are connected or not. Recall there are $N(N-1)/2$ pairs of integers i, j in $\{1, ..., N\}$ with $i \leq j$. Consider the set of functions $f : \boldsymbol{R}^{N(N-1)/2\times 2} \to \boldsymbol{R}^{N(N-1)/2\times 2}$ that

are equivariant to the permutations of columns (the vectors $((i,j),\sigma_{i,j})$), invariant under permutations of the labelling of the nodes of the graph and distinguish non-isomorphic graphs. Pick one such f, then theorem 1 says the function f can be approximated with arbitrary accuracy by Transformers taking as input the sequences $((i,j),\sigma_{i,j})_{i\leq j\in\{1,\ldots,N\}}$.

Now we consider the problem of classification of graph isomorphism classes using Transformers on the nodes. Choosing instead as input the sequence of graph nodes with positional encoding the columns of the adjacency matrix and $f : \boldsymbol{R}^{N\times N} \to \boldsymbol{R}^{N\times N}$ a map that is invariant under permutation of node order and has distinct values for different isomorphism class of graphs, theorem 2 says that the map f can be approximated by Transformers in the L^p distance with arbitrary accuracy.

This may seem strange since it is unlikely there is an algorithm solving the graph isomorphism problem in polynomial time to the number of nodes N, and we address this issue in the notes below.

Note 1: Only an approximate solution. The universality theorems do not state that Transformers solve the isomorphism problem, but that they can approximate a solution. They only learn the invariant functions only up to some error so they still can mislabel graphs.

Note 2: Estimate of number of Transformer blocks. For the approximation of the function f by a Transformer to be precise, a large number of Transformer blocks will be needed. In [56], it is stated that the universal class of function is obtained by composing Transformer blocks with 2 heads of size 1 followed by a feed-forward layer with 4 hidden nodes. In [57] section 4.1, an estimate of the number of blocks is given. If $f : \boldsymbol{R}^{d\times n} \to \boldsymbol{R}^{d\times n}$ is L-Lipschitz, then $||f(\boldsymbol{X}) - f(\boldsymbol{Y})|| < \varepsilon/2$ when $||\boldsymbol{X} - \boldsymbol{Y}|| < \varepsilon/2L = \delta$. In the notation of [57], the LPE has constants $p = 2$ and $s = 1$. If g is a composition of Transformer blocks then an error $||f - g||_{L^p} < \varepsilon$ can be achieved with a number of Transformer blocks larger than

$$\left(\frac{dn}{\delta}\right) + \left(\frac{p(n-1)}{\delta^d} + s\right) + \left(\frac{n}{\delta^{dn}}\right) = \frac{dn2L}{\varepsilon} + \frac{2(n-1)(2L)^d}{\varepsilon^d} + 1 + \frac{n(2L)^{dn}}{\varepsilon^{dn}}$$

In the case of the node encoder described above, $n = d = N$ (the number of nodes) and the last term in the sum above becomes $N(2L/\varepsilon)^{N^2}$, so the number of parameters and therefore the computational time is exponential in the number of nodes for a fixed error ε. Note that this bound on the number of Transformer blocks might not be tight and might be much lower for a specific problem.

Note 3: Learning invariance to label permutations. In the above proof, the Transformer is assumed to be able to label all isomorphic graphs into the same class within a small error. However, given a graph of N nodes, there are $N!$ different node labeling permutations, and they all need to be mapped to the same output class. It seems unlikely that such function can be learned with polynomial complexity to N.

Following these observations, it does not seem appropriate to compare Transformers to the WL test as is the custom for graph neural networks and we think at this point we should seek a new measure of expressiveness of graph Transformers.

C.3 Expressivity of the node-LPE

Here, we want to show that the proposed node-LPE can generate a unique node identifier that allows our Transformer model to be a universal approximator on graphs, thus allowing us to approximate a solution to graph isomorphism.

Recall the node LPE takes as input an $N \times m \times 2$ tensor with m the number of eigenvalues and eigenvectors that are used to represent the nodes. The output is a $N \times k$ tensor. Notice that 2 non-isomorphic graphs on N nodes can have the same $m < N$ eigenvalues and eigenspaces and disagree on the last $N - m$ eigenvalues and eigenspaces. Any learning algorithm missing the last $N - m$ pieces of information won't be able to distinguish these graphs. Here we will fix some m and show that the resulting Transformer can approximately classify all graphs with $N \leq m$.

Fix some linear injection $\boldsymbol{M} : \boldsymbol{R}^{N\times 2\times m} \to \boldsymbol{R}^{N\times k\times m}$. Let G be a graph and $U \subset \boldsymbol{R}^{N\times 2\times m}$ be bounded set containing all the tensor representations of graphs $\boldsymbol{T}_G$ and let R be the

radius of a ball containing $\boldsymbol{M}(U)$. Consider the set $C^0(\boldsymbol{B}_R^{N\times k\times m}, \boldsymbol{R}^{N\times k\times m})$ of continuous functions of the closed radius R ball in $\boldsymbol{R}^{N\times k\times m}$. Finally, denote by $\boldsymbol{S} : \boldsymbol{R}^{N\times k\times m} \to \boldsymbol{R}^{N\times k}$ the linear function taking the sum of all values in the m dimension. The following universality result for LPE Transformers is a direct consequence of theorem 2.

Proposition 1 *For any $1 \leq p < \infty$, $\varepsilon > 0$ and any continuous function $F : \boldsymbol{B}_R^{N\times k\times m} \to \boldsymbol{R}^{N\times k}$, there is an LPE Transformer g such that the L^p distance between $M \circ f \circ S$ and g is smaller than ε.*

As a corollary, we get the same kind of approximation to solutions of the graph isomorphism problem as with the naive Transformers. Let f be a function of $C^0(\boldsymbol{B}_R^{N\times k\times m}, \boldsymbol{R}^{N\times k\times m})$ that maps $M(\boldsymbol{T}_G)$ to a value that is only dependent of the isomorphism class of the graph and assigns different values to different isomorphism classes. We can further assume that f takes values that are 0 for all but one coordinate in the k dimension. The same type of argument is possible for the edge-LPE from figure A.1.

C.4 Comparison of the learning complexity of naive graph Transformers and LPE

We now argue that while the LPE Transformer and the naive graph Transformers of section C.2 can all approximate a function f solution of the graph isomorphism problem, the complexity of the learning problem of the LPE is much lower since the spaces it has to learn are simpler.

Naive Node Transformer. First recall that the naive node Transformer learns a map $f : \boldsymbol{R}^{N^2} \to \boldsymbol{R}^{N^2}$. In this situation, each graph is represented by $N!$ different matrices which all have to be identified by the Transformer. This encoding also does not provide any high-level structural information about the graph.

Naive Edge Transformer. The naive edge Transformer has the same difficulty since the function its learning is $\boldsymbol{R}^{N(N-1)} \to \boldsymbol{R}^{N(N-1)}$ and the representation of each edge depend

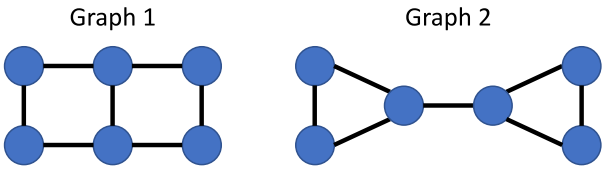
on a choice of labeling of the vertices and the $N!$ possible labelings need to be identified again.

Node-LPE Transformer. In the absence of eigenvalues with multiplicity > 1, the node LPE that learns a function $\boldsymbol{R}^{N\times 2\times m} \rightarrow \boldsymbol{R}^{N\times k}$ does not take as input a representation of the graph that depends on the ordering of the nodes. It does, however, depend on the choice of the sign of each of the eigenvectors so there are still 2^N possible choices of graph representations that need to be identified by the Transformer but this is a big drop in complexity compared to the previous $N!$. The eigenfunctions also provide high-level structural information about the graph that can simplify the learning task of the graph.

Edge-LPE Transformer. Finally, the edge LPE of appendix A uses a graph representation as input that is also independent of the sign choice of the eigenvectors so each graph has a unique representation (considering the absence of eigenvalues with multiplicity > 1). Again, the eigenfunctions provide high-level structural information that is not available to the naive Transformer.

LPE Transformers for non-isospectral graphs. Isospectral graphs are graphs that have the same set of eigenvalues despite having different eigenvectors. Here, we argue that the proposed node LPE can approximate a solution to the graph isomorphism problem for all pairs of non-isospectral graphs, without having to learn invariance to the sign of their eigenvectors nor their multiplicities. By considering only the eigenvalues in the initial linear layer (assigning a weight of 0 to all ϕ), and knowing that the eigenvalues are provided as inputs, the model can effectively learn to replicate the input eigenvalues at its output, thus discriminating between all pairs of non-isospectral graphs. Hence, the problem of learning an invariant mapping to the sign of eigenvectors and multiplicities is limited only to non-isospectral graphs. Knowing that the ratio of isospectral graphs decreases as the number of nodes increases (and is believed to tend to 0) [47], this is especially important for large graphs and mitigates the problem of having to learn to identify 2^N with eigenvectors with different signs. In Figure C.1, we present an example

of non-isomorphic graphs that can be distinguished by their eigenvalues but not by the 1-WL test.



Eigenvalues of Graph 1	Eigenvalues of Graph 2
0	0
1	0.438
2	3
3	3
3	3
5	4.562

Figure C.1: Example of non-isomorphic non-isospectral graphs that can be distinguished by the eigenvalues of their Laplacian matrix, but not by the 1-WL test.