ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ECT-based Positional Encoding for Graph Neural Networks

Master Thesis

J. P. Garcia Amboage

September 15, 2025

Advisors: Prof. Dr. B. Gärtner, Prof. Dr. B. Rieck, Dr. P. Schnider
Department of Computer Science, ETH Zürich

Abstract

Graph learning problems are commonly approached using Message Passing Neural Networks (MPNNs). Yet, MPNNs face well-known limitations such as oversmoothing, oversquashing, and restricted expressivity. Graph positional encodings (PEs) have been proposed to mitigate these issues by injecting structural information into MPNNs as well as into alternative architectures that would otherwise be structure-agnostic. The Euler Characteristic Transform (ECT), a tool from topological data analysis, characterizes shapes by tracking the evolution of their Euler characteristic across multiple directions. In this thesis, we introduce LEAP, a novel graph PE based on the local ECT (l-ECT). We integrate LEAP into graph neural architectures in an end-to-end trainable manner. Experiments on real-world and synthetic datasets show that LEAP consistently outperforms established baselines and captures topological features in tasks where standard MPNNs struggle. These findings highlight the potential of LEAP as a powerful tool for graph learning problems.

Contents

Chapter 1

Introduction

In multiple domains such as high energy physics, biomedicine, and social sciences, data can often be naturally represented as *graphs* [35, 4, 37, 46, 44]. As a result, many prediction problems in these areas are tackled using *Graph Neural Networks* (GNNs) [11, 14]. By iteratively propagating and aggregating information across nodes and edges, a process known as *message passing*, GNNs have achieved strong performance on both node-level and graph-level prediction tasks.

Despite their success, standard message-passing architectures still face important limitations [45, 6, 15, 43]. From a theoretical perspective, conventional message-passing models such as GCNs have been proven to be at most as expressive as the 1-dimensional Weisfeiler–Lehman test [24, 6]. From a practical perspective, message-passing architectures often suffer from *oversmoothing* [33, 48] and *oversquashing* [8], which limit their ability to assign distinct hidden representations to nearby nodes and to capture long-range dependencies in the graph.

In recent years, several GNN variants have been developed to overcome these shortcomings. Some approaches focus on modifying the traditional message-passing paradigm to achieve provably stronger expressivity bounds, as in the case of GIN [45]. Other approaches alter the original graph structure by adding virtual nodes or edges to improve information flow through the model [1]. A particularly prominent direction, inspired by the success of Transformer models in Natural Language Processing (NLP), consists in combining message passing with global attention mechanisms to capture long-range dependencies more effectively [27, 3, 34].

In this context, positional encoding (PE) for graphs [27, 15, 11] arises as a mechanism to enrich node features with information about a node's role within the graph or its local neighborhood. Graph PEs are particularly relevant when working with architectures that do not directly account for

graph structure, as it allows them to leverage connectivity information that may be crucial for the task at hand [27]. Moreover, the use of graph PEs has been shown to enhance performance even for conventional message-passing GNNs [20, 12].

The concept of node position in a graph admits several interpretations. For example, even a simple feature such as node degree may already be regarded as a PE, since it provides information about a node's local neighborhood. For this reason, multiple types of graph PEs have been proposed. One common strategy is to derive static encodings from structural properties of the graph, such as the adjacency or degree matrices, and use them as additional node input features [11, 15]. Other approaches, such as [12], introduce learnable PEs that are updated by the GNN at each message-passing step, after being initialized from a static encoding. More recently, [42] proposed incorporating topological information via persistent homology computed on top of PE features. Since different graph PEs capture different notions of position, some approaches, such as [27], combine multiple graph PEs rather than relying on a single one.

The Euler Characteristic Transform (ECT) [38, 25, 28] is a tool from topological data analysis that tracks how the Euler characteristic of a shape evolves across multiple directions. The ECT has been shown to be injective for broad classes of shapes [13, 7]. Although the ECT itself is not differentiable, differentiable approximations have been developed, enabling its use in end-to-end trainable neural architectures [30, 31]. These approaches have demonstrated strong potential in applications such as point cloud synthesis and graph learning. Furthermore, the local ECT (l-ECT) [43] has been introduced as a means of generating node-level features. However, existing work has used it only to precompute static, handcrafted node features.

A key advantage of the ECT is that it can be computed efficiently [30], unlike persistent homology, another topological descriptor that has already been integrated with GNNs [16, 41] but remains computationally expensive, limiting its scalability.

Inspired by the success of the ECT and its variants in various learning tasks, we propose *LEAP PE* (l-ECT and Projection Positional Encoding), a novel learnable PE scheme for graphs. LEAP integrates the l-ECT into graph deep learning architectures in an end-to-end trainable manner, enabling the directions along which the transform is computed to be optimized jointly with the model. In addition, LEAP can be applied not only to raw node features but also to learned intermediate representations, allowing the encoding to adapt dynamically during training.

More precisely, our main contributions are as follows:

- *First end-to-end trainable use of the l-ECT.* To the best of our knowledge,

this is the first work to integrate the l-ECT into deep learning architectures, enabling learnable directions and compatibility with learned features.

- *Projection strategies for ECT/l-ECT.* We introduce and evaluate several projection mechanisms to transform the discretized l-ECT representations into fixed-dimensional vectors of lower dimension, with designs that explicitly account for permutation invariance with respect to the directions in which the ECT is computed.
- *Comprehensive evaluation.* We benchmark our approach on eight real-world graph datasets in combination with different graph learning architectures, and compare it against established positional encodings such as LaPE and RWPE. Moreover, we introduce a synthetic dataset specifically designed to test the ability of our method to capture topological features.

Our empirical results show that the l-ECT can be effectively leveraged as a PE for graph learning problems. Moreover, in most of the evaluated tasks, *LEAP* outperforms all considered baselines, demonstrating its ability to learn embeddings that capture useful structural information about graphs. These findings open the door to combining the *LEAP* with other graph PEs and integrating it into more complex architectures.

The remainder of this thesis is organized as follows. In Chapter 2, we review the necessary background on graph neural networks, positional encodings, and the Euler Characteristic Transform. In Chapter 3 we present our proposed l-ECT–based PE method and discuss its limitations. In Chapter 4, we describe the experimental setup and datasets. Chapter 5 reports and discusses the results. Finally, Chapter 6 concludes with a summary and directions for future work.

Chapter 2

Background

In this chapter, we present fundamental notions of *graph learning*, including common architectures and positional encodings. We then formally introduce the Euler Characteristic Transform (ECT) [38], along with its differentiable approximation (DECT) [30] and local variant (l-ECT) [43]. These are the key concepts for understanding our proposal of a new Positional Encoding strategy for graph learning problems based on the l-ECT.

2.1 Graph Learning with Neural Networks

Before talking about *graph learning*, we first define what we mean by a *graph* and by an specific class of graphs that we will call *featured graphs* which hold particular relevance in this domain.

Definition 2.1 *An* undirected graph *is a pair $G = (V, E)$, where:*

- *V is a finite set of elements called* nodes *or* vertices.
- *$E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ is a set of unordered pairs of distinct nodes called* edges.

Through out the text we may use $V(G)$ and $E(G)$ respectively to refer to the nodes and edges of a given graph G. Moreover, unless stated otherwise, the term graph *will be used to mean* undirected graph.

Definition 2.2 *A* featured graph *is a pair (G, x), where G is an undirected graph, and x is a map $x : V(G) \to \mathbb{R}^n$. For a given node $v \in V(G)$, we refer to $x(v)$ as the* features *of v.*

As a simple example of a graph, one can consider a train network where nodes correspond to train stations and an edge is present between two nodes if there is a direct connection between the corresponding stations. This graph

can be extended to a featured graph by assigning to each station v a feature vector $x(v) \in \mathbb{R}^2$ representing its geographic coordinates.

By *graph learning*, we refer to learning problems in which the input data consists of graphs. In this setting, each data sample is a graph, and our goal is to predict a regression or classification target for each of them. When predictions concern the entire graph, the problem is referred to as a *graph-level task*, whereas when predictions are made for individual nodes within a graph, it is called a *node-level task*.

An example of a regression graph-level task is one in which we are given a set of molecules, each represented as a graph where the atoms are nodes and the edges represent bonds between them, and the goal is to predict the melting temperature of each molecule. An example of a classification node-level task is a problem where, given a set of molecules (again, represented as graphs), we aim to predict, for each node (atom) in a given molecule, its chemical element (for example, carbon, oxygen, etc.).

In general, a graph learning problem consists of a training set of graphs along with their corresponding labels. A model is then trained (or, equivalently, optimized) to predict the correct labels for the given graphs, with the aim of generalizing to unseen graphs drawn from the same distribution as the training set. Typically, the graphs in a given problem can vary in the number of nodes and edges they contain, and there is no inherent ordering of the nodes.

We will focus on learning problems on featured graphs. In this case, the node features across all the graphs in the problem have the same dimension.

2.1.1 Message Passing Neural Networks

The Multi-Layer Perceptron (MLP) [32], a fundamental building block of deep learning architectures, acts as a learnable mapping $\text{MLP} : \mathbb{R}^n \to \mathbb{R}^m$. Therefore, it is well suited for learning problems where each data sample can be naturally represented as a fixed-size vector $p \in \mathbb{R}^n$. In contrast, the MLP is not directly applicable to graph-structured data as, in general graph learning problems, we have that:

- The number of nodes and edges varies across graphs in the dataset.
- There is not a canonical node ordering.
- The connectivity patterns of the graphs may contain information relevant for the task at hand.

Message Passing Neural Networks (MPNNs) [14, 18], are a family of deep learning architectures designed to meet these requirements. MPNNs can handle unordered sets of nodes of varying sizes and capture their connectivity

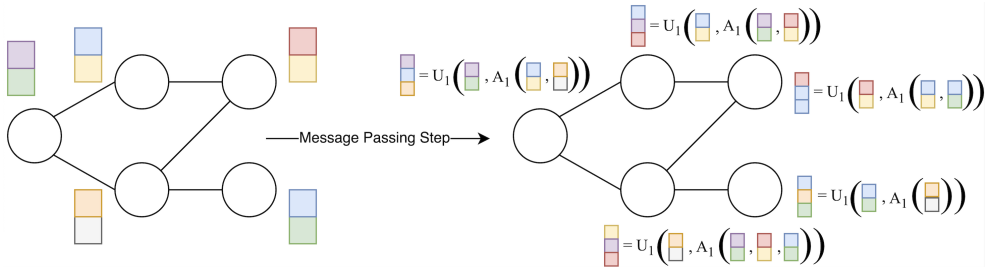


Figure 2.1: Representation of the first message passing step in a MPNN. $\mathrm{A}_1(\cdot)$, and $\mathrm{U}_1(\cdot)$ represent the update and aggregation maps from Equation 2.1. Each node gets an updated feature vector that is computed as a function of its previous feature vector, and the feature vectors of its neighboring nodes.

(the graph structure) by iteratively propagating information along the graph edges.

In particular, a given MPNN has a fixed number of steps T, and, for each step $t \in \{0, \ldots, T\}$, there is a hidden state $h_v^{(t)} \in \mathbb{R}$ for each node v in the graph that is being processed. The initial state of each vertex v corresponds to its features $x(v)$, or to a learnable transformation of them. The rest of the states are computed according to the update rule

$$h_v^{(t+1)} = \mathrm{UPDATE}_{t+1}\left(h_v^{(t)}, \mathrm{AGGREGATE}_{t+1}\left(\{h_u^{(t)}, u \in \mathcal{N}(v)\}\right)\right), \quad (2.1)$$

where UPDATE_t is a learnable map (i.e. an MLP applied to the concatenation of h_v^t and the result of $\mathrm{AGGREGATE}_t$), $\mathrm{AGGREGATE}_t$ is a permutation invariant learnable map that can handle inputs of different sizes (i.e. an MLP applied to the sum of the elements in its input set), and $\mathcal{N}(v)$ is the set of nodes neighboring the node v.

For node-level tasks, the same learnable map (typically an MLP) is applied to the final state of each of the nodes to make node-level predictions. For graph-level tasks, the final hidden states of all the nodes in the graph are aggregated together using a *pooling layer* (i.e. sum, mean, max-pooling...) [19]. The aggregation produces a fixed-size learned representation of the entire graph. Finally, a learnable map is applied to this graph representation to make graph-level predictions. Since all the layers use parameterized, differentiable operations, the entire architecture can be trained end-to-end for both node-level and graph-level tasks.

There are multiple MPNNs architectures depending on the choice of the UPDATE and AGGREGATE functions. We will briefly introduce two popular ones: Graph Convolution Networks (GCNs) [18], and Graph Attention Networks (GATs) [40], as these are the ones used for the experiments conducted in this work.

More precisely, in GCNs the operations for updating and aggregating the node states in Equation 2.1 are defined as:

$$\text{AGGREGATE}_{t+1}\left(\{h_u^{(t)}, u \in \mathcal{N}(v)\}\right) := \sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{\hat{d}_u \hat{d}_v}} h_u^{(t)},$$
$$\text{UPDATE}_{t+1}\left(h_v^{(t)}, w\right) := \varphi_t\left(W_t^\top (\frac{1}{\hat{d}_v} h_v^{(t)} + w)\right),$$

where $\hat{d}_i := 1 + |\mathcal{N}(i)|$, $W_t \in \mathbb{R}^{d_t \times d_{t+1}}$ is a matrix whose entries are learned, and φ_t is a non linear activation function (i.e. ReLU). The normalization terms $\hat{d}_u$ help to prevent overweighting the information from the more connected nodes in the graph.

When it comes to GATs the operations for updating and aggregating the node states are defined as:

$$\text{AGGREGATE}_{t+1}\left(\{h_u^{(t)}, u \in \mathcal{N}(v)\}\right) := \sum_{u \in \mathcal{N}(v)} \alpha_{v,u}^{(t)} h_u^{(t)},$$
$$\text{UPDATE}_{t+1}\left(h_v^{(t)}, w\right) := \varphi_t\left(W_t^\top (\alpha_{v,v}^{(t)} h_v^{(t)} + w)\right),$$

where W_t is a matrix whose entries are learned, φ_t is a non linear activation function, and the coefficients $\alpha_{u,v}^{(t)}$ are referred to as the *attention coefficients* and are computed as:

$$\alpha_{u,v}^{(t)} := \frac{\exp\left(\text{LeakyReLu}(a_t^\top [M_t h_v || M_t h_u])\right)}{\sum_{k \in \mathcal{N}(v) \cup \{v\}} \exp\left(\text{LeakyReLu}(a_t^\top [M_t h_v || M_t h_k])\right)},$$

where $\text{LeakyReLU}(x) := \max(0, x) + s \cdot \min(0, x)$, with $s \in \mathbb{R}$ being a hyper-parameter of the function, $[\,\cdot\,||\,\cdot\,]$ denotes the concatenation of vectors, M_t is a matrix with learnable entries, and a_t is a vector with learnable entries.

The GAT architecture uses a variation of *attention* [39] to aggregate the node states in a learnable way. Attention is is one of the key components of the Transformer [39] architecture, a family of models originally designed for Natural Language Processing (NLP) tasks. Inspired by its success on NLP tasks, GATs allow attention to be used for neighborhood aggregation graph learning problems.

Note that in the GAT architecture is possible to have multiple *attention heads*. This means that several independent attention mechanisms are applied in a given step of the neural network, and their outputs are then concatenated or averaged to form the node states used in the next step of the network. Also, both GCNs and GATs can be extended to handle graphs where the edges are also "featured" .

MPNNs have been successfully applied in graph learning problems across multiple domains such as biochemistry, high energy physics, or recommender systems [14, 35, 46]. However, the most common architectures such as GCNs and GATs suffer from several practical and theoretical limitations [43, 11, 15, 34, 12, 2].

In particular, two well studied phenomena that tend to happen when working with these architectures are:

- *Oversmoothing* [33, 48]: nodes that are close in the graph tend to end up having hidden states indistinguishable from one another even if they are different from the point of view of the task at hand.
- *Oversquashing* [8]: relevant information from distant nodes fails to propagate through the network which results in the model failing to capture long-range dependencies in the graph.

From a theoretical point of view, it has been shown that "standard" MPNNs architectures, such GCNs or GATs among others, cannot be more expressive than the 1-dimensional Weisfeiler-Leman (1-WL) test [24, 45]. More precisely, this implies that these models fail to distinguish between particular types of non-isomorphic graphs, and that they cannot perform certain subgraph counting tasks [43, 6].

2.1.2 Transformers

Before discussing Positional Encoding for graphs, we briefly introduce the *Transformer architecture* [39], since positional encodings were originally proposed together with this architecture as a mechanism to incorporate information about the positions of tokens[1] in a sequence. Moreover, Transformers have also inspired multiple architectures designed to address the limitations of MPNNs [40, 27, 34].

Transformers were first introduced for dealing with Natural Language Processing (NLP) tasks and remain the backbone of modern large language models (LLMs) such as GPT architectures [26]. They are designed to process variable-length sequences, and the original architecture had two main components: an *encoder* and a *decoder*.

The encoder maps an input sequence into a hidden representation for each token, where each representation captures not only the content of the token itself but also its relation to the other tokens in the sequence. The decoder then uses these hidden representations to generate new sequences, for example in

[1] Tokens are the atomic unit for NLP models, common words typically are represented by their own token and more complex or rare words can be represented by a combination of multiple tokens.

machine translation or text completion. The core operation in both encoder and decoder is the *attention mechanism*.

Given three matrices $Q \in \mathbb{R}^{n\times d}$, $K \in \mathbb{R}^{n\times d}$, and $V \in \mathbb{R}^{n\times p}$, the *attention* operation is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V, \tag{2.2}$$

where softmax is applied row-wise in the matrix $QK^\top$, and we have that softmax : $\mathbb{R}^n \to \mathbb{R}^n$, is defined as $\text{softmax}(v)_i = \exp(v_i)/\sum_{j=0}^{n}\exp(v_j)$ for all $v \in \mathbb{R}^n$.

Usually, the matrices Q, K, and V are referred to as queries, keys and values respectively. This results in a nice interpretation of the attention operation as a "soft" database lookup. We can interpret Q as collection of d-dimensional vectors representing n queries to a database. In the database there are also n values stored, which are the p-dimensional vectors that form the matrix V. For each of these values, we have a d-dimensional key, these are the rows of the matrix K. Thus, with the attention operation we retrieve a vector for each of the n queries in Q, each of this vectors is a weighted average of the values in V, where the weight of each value corresponds to how well its key matched the query vector.

The encoder of the Transformer architecture uses a particular version of the attention operation named *self-attention*. Given a sequence of input vectors $x_1, \ldots, x_n$, aranged as a matrix $X \in \mathbb{R}^{n\times k}$, *self-attention* of X is just attention computed on the matrices $Q = XW_Q$, $K = XW_K$, and $V = XW_V$, where W_Q, W_K, and W_V are learnable matrices. Therefore, using self-attention results in a hidden representation of each vector in the input sequence that takes into account how that vector relates to all the other elements in the same sequence.

Remark 2.3 *Changing the order of the vectors $x_1, \ldots, x_n$ in the input sequence of a Transformer encoder does not change their hidden representations computed with self-attention.*

Reordering the input vectors $x_1, \ldots, x_n$, results in a reordering of the rows in the matrix X. The reordered version of X can be expressed as $R_\pi X$, where R_π is a row permutation matrix.

This results in new query, key and value matrices that we can denote by Q', K' and V'. In particular we have that:

$$Q' = (R_\pi X)W_Q = R_\pi Q, \;\; K' = (R_\pi X)W_K = R_\pi K, \;\; V' = (R_\pi X)W_V = R_\pi V.$$

Then, we get that $Q'(K')^\top = (R_\pi Q)(R_\pi V)^\top = R_\pi(QK^\top)R_\pi^\top$. *Since* $\mathrm{softmax}(\cdot)$ *is applied row-wise, we also get that*

$$\mathrm{softmax}(R_\pi(QK^\top)R_\pi^\top) = R_\pi \mathrm{softmax}((QK^\top)R_\pi^\top).$$

From the definition of $\mathrm{softmax}(\cdot)$*, it follows that reordering the coordinates of a vector permutes with the application of* $\mathrm{softmax}(\cdot)$*. Therefore,*

$$R_\pi \mathrm{softmax}((QK^\top)R_\pi^\top) = R_\pi \mathrm{softmax}(QK^\top)R_\pi^\top$$

.

Hence, we get that $\mathrm{Attention}(Q', K', V') = (R_\pi \mathrm{softmax}(QK^\top/\sqrt{d})R_\pi^\top)(R_\pi V)$. *Since permutation matrices are orthonormal, the previous expression can be simplified to* $R_\pi \mathrm{softmax}(QK^\top/\sqrt{d})V = R_\pi \mathrm{Attention}(Q, K, V)$.

This means that, without any additional information, the transformer encoder is permutation equivariant with respect to the input sequence. That is, the position of the vectors in the input sequence is not taken into account when computing their hidden representations. Therefore, the sequence structure is disregarded, and the input is effectively being processed as a set of vectors.

Since the positions of words in a sentence can be relevant information that the model would otherwise ignore[2], the authors of the Transformer architecture proposed to use positional encodings as a mechanism to enhance the input vectors $x_1, \ldots, x_n$ with information about their position in the sequence. More precisely, they introduced *sinusoidal positional encodings* [39], these are vectors computed from the position of each element in the input sequence, which are then added to the corresponding input vector x_i before feeding the sequence into the self-attention mechanism. Beyond sinusoidal encodings, many other types of sequence positional encodings have been proposed, such as *RoPE* [36], which applies position-dependent rotations to the coordinates of x_i.

The success of the Transformer architecture has led to its adaptation to domains beyond NLP, such as computer vision [10], or, as we have already mentioned, graph learning [27, 40, 34]. In this context, a Transformer encoder can generate hidden representations of each node that directly "attend" to all other nodes, thereby offering potential to overcome some of the limitations of MPNNs, such as oversquashing.

2.1.3 Graph Positional Encoding

Several approaches have been proposed to overcome the limitations of MPNNs outlined at the end of Subsection 2.1.1. Some of these include altering the original connectivity of the graphs, for example by introducing

[2]Word order can be crucial in natural language, as it is not the same to say "I like dogs but I hate cats" as it is to say "I like cats but I hate dogs".

a *virtual node* connected to every other node with the aim of improving information flow [1, 15]. Some other approaches like GPS [27] use message passing followed by a Transformer, allowing each node to directly look at every other node in the graph.

In these approaches, the graph structure is either modified or not explicitly encoded into the model. Within this context, graph Positional Encodings (PEs) [15, 27] emerge as a mechanism to inject structural information into the node features, enabling the models to remain aware of the underlying graph topology. This plays a role analogous to sequence positional encodings in NLP, where they are used to make Transformers aware of the positions of tokens in a sequence.

However, when dealing with graphs rather than sequences, there is no straightforward notion of a node's "position". For this reason, a wide variety of graph positional encoding strategies have been proposed, including even learnable ones. The work by [27] proposes a categorization of the graph PE strategies dividing them into *Positional Encodings* (PEs) and *Structural Encodings* (SEs), and also on whether they are *local*, *global* or *relative*.

For example, the distance between a node and the centroid of a cluster containing that node would be a *Local PE*. If the distance is instead computed with respect to the centroid of the graph we would be talking about a *Global PE*. The degree of a node could act as a basic *Local SE* as it is information related to the structure of the local neighborhood around the node. When it comes to *Global SE*, in [27] they are considered graph features instead of node features. Some of the provided examples are graph properties such as its diameter, number of connected components etc. Lastly, both *Relative PE* and *Relative SE* are considered as additional edge features as they directly capture information about the relation between two nodes.

Besides this categorization, multiple authors simply use the term PE to refer to any additional node features that contain information about the positional or structural role of a node in the graph [15, 12]. Throughout our work, we will adopt this usage of the term PE. However, when appropriate, we will distinguish between positional/structural and global/local PEs to provide the reader with a clearer understanding of the type of information captured by a given PE.

It is also is worth noting that, unlike in NLP, where positional encodings are typically added to the input vectors, in graph learning it is common to concatenate the positional encoding to the original node features [11], this allows the model to be aware of which information comes from the original node features and which from the positional encoding. Moreover, this also allows to use PEs of different dimensionality than that of original node features.

We now introduce two graph PEs particularly relevant in the literature [11, 15, 22, 12], that are also used for the experiments of this work.

Random Walk PE (RWPE) Given a graph G and a node $v \in V(G)$, [12] define the k-dimensional RWPE of v, denoted by $p_v^{\text{RWPE}_k}$ as:

$$p_v^{\text{RWPE}_k} := [\mathbf{RW}_{vv}, (\mathbf{RW})^2_{vv}, \ldots, (\mathbf{RW})^k_{vv}] \in \mathbb{R}^k,$$

where $\mathbf{RW} := A(G)D(G)^{-1}$ is the random walk matrix of the graph G, $A(G)$ denotes the *adjacency matrix* of G, and $D(G)$ denotes the *degree matrix* of G.

Laplacian PE (LaPE) Given a graph G, its *normalized Laplacian matrix* is given by $L(G) = I - D(G)^{-1/2}A(G)D(G)^{-1/2}$, where I stands for the identity matrix. The LaPE of the nodes in G are constructed from the eigendecomposition of $L(G) = Q^\top AQ$. Given the eigenvalues sorted in ascending order $\lambda^{(1)}, \ldots, \lambda^{(K)}$, and their corresponding eigenvectors $q^{(1)}, \ldots, q^{(K)}$, [11] define the k-dimensional LaPE of a node $v \in V(G)$, denoted by $p_v^{\text{LaPE}_k}$ as:

$$p_v^{\text{LaPE}_k} := [q_v^{(i)}, q_v^{(i+1)}, \ldots, q_v^{(i+k)}] \in \mathbb{R}^k,$$

where i is the index of the first non-trivial eigenvector.

Under the categorization proposed by [27], RWPE is considered a local structural encoding, whereas LaPE is regarded as a global positional encoding. A number of variants of both of these of PEs have been proposed [15, 22].

Finally, we highlight that, as we have already mentioned before, empirical evidence has shown that incorporating graph PEs can be beneficial not only in settings with modified architectures or artificially altered graph connectivity, but also for improving the performance of standard MPNNs when used as additional node features [11, 20, 12].

2.2 Euler Characteristic Transform

Before talking about the Euler Characteristic Transform, we should first introduce the Euler characteristic.

Definition 2.4 *Given a simplicial complex K its* Euler characteristic $\chi(K)$ *can be defined as*

$$\chi(K) := \sum_{i=0}^{d} (-1)^i |K^{(i)}|,$$

where d is the maximum dimension of any simplex in K, and $K^{(i)}$ denotes the set of the i-dimensional simplices in K.

Note that there are several equivalent alternative definitions of the Euler characteristic [28]. For example, in terms of the *Betti numbers* of the simplicial complex. However, the definition above, where the Euler characteristic is built as an alternating sum of the number of simplices of each dimension, was chosen as it is easy to follow even for those with a more limited background in topology. Hence, highlighting the simplicity of the Euler characteristic.

As an example, the Euler characteristic of a graph, which can be thought as a 1-dimensional simplicial complex, is just the number of nodes in the graph minus the number of edges. Similarly, the Euler characteristic of a polyhedron can be computed as the number of vertices minus the number of edges plus the number of faces.

Following from the previous example, one can play around with the Platonic solids, a well known family of polyhedrons, after computing their Euler characteristic one would realize that all of them have the same Euler characteristic which is 2. This is not a coincidence, it happens because all the Platonic solids are homeomorphic to the sphere and the Euler characteristic is a *topological invariant* [25, 28]. That is, homotopic topological spaces have the same Euler characteristic[3].

Thanks to this invariance property, the definition of the Euler characteristic extends naturally to all triangulable topological spaces [25][4]. Moreover, the Gauss–Bonnet Theorem [9] shows that the Euler characteristic of any compact two-dimensional Riemannian manifold can be expressed in terms of its curvature, In this way, the theorem bridges topology and geometry by relating a topological invariant to a differential-geometric quantity.

As we have seen, the Euler characteristic, despite its simplicity, has some remarkable properties. In particular, given two topological spaces, if their Euler characteristics differ, we can immediately conclude that they are not equivalent from a topological perspective. A natural question then arises: does the converse hold? That is, if two spaces have the same Euler characteristic, can we conclude that they are homotopic? The answer is no: two spaces may share the same Euler characteristic without being topologically equivalent. This phenomenon is illustrated in Figure 2.2, where the two graphs depicted have the same Euler characteristic but differ in the number of connected components, and hence cannot be homotopic.

[3]This directly follows from the expression of the Euler characteristic in terms of Betti numbers.

[4]Since all triangulations of a triangulable topological space are homotopic they have the same Euler characteristic, thus we can define this as the Euler characteristic of the space itself.

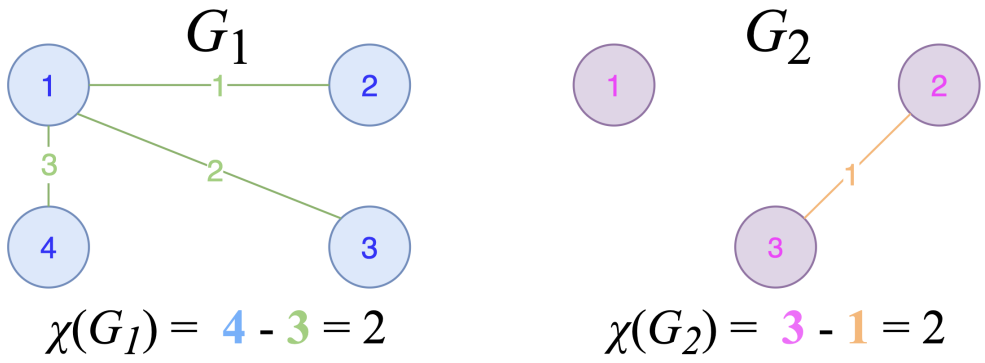


Figure 2.2: Two graphs with the same Euler characteristic, and with a different number of connected components. The Euler characteristic is computed as the number of nodes minus number of edges as graphs are are 1-dimensional simplicial complexes.

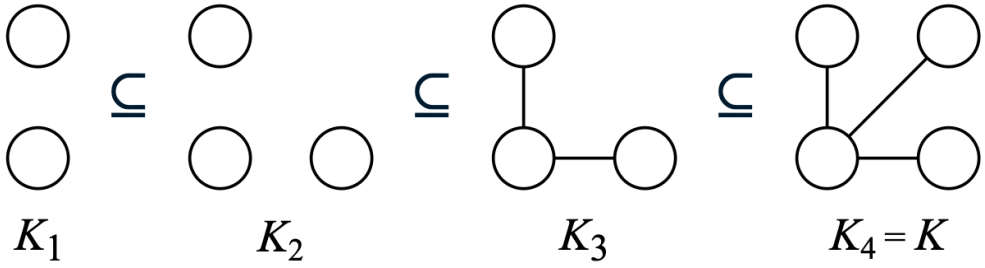


Figure 2.3: Example of a filtration in a 1-dimensional simplicial complex K. Each K_i is a valid simplicial complex (no edge is added before the involved nodes are added), and $K_i \subseteq K_j$ whenever $i \leq j$.

2.2.1 Introduction to the ECT

Although the Euler characteristic alone cannot always distinguish between non-homotopic topological spaces, it can be used to construct a more powerful descriptor, the Euler Characteristic Transform (ECT) [38], which is injective on certain classes of topological spaces [13, 25, 28].

The underlying idea behind the ECT is to track how the Euler characteristic of a simplicial complex evolves as the complex is grows in every possible direction. Before formally presenting the ECT, we first introduce several concepts that will allow us to build an intuitive definition.

Definition 2.5 *Given a simplicial complex K, a* filtration *of K is defined as an indexed nested collection of simplicial complexes $K_i \subseteq K$ with $i \in I \subseteq \mathbb{R}$, such that for any $a, b \in I$ with $a \leq b$ it holds that $K_a \subseteq K_b$.*

In Figure 2.3 we show an example of a filtration in a 1-dimensional simplicial complex, which can just be regarded as a graph. The filtration in the figure is indexed by the set $I = \{1, 2, 3, 4\}$. Note, however, that in general the index set does not need to be finite.

Remark 2.6 *Given a simplicial complex K, any real valued map $f : K^{(0)} \to \mathbb{R}$ can be used to induce a filtration on K. Here, following the notation introduced in Definition 2.4, $K^{(0)}$ denotes the set of 0-simplices of K, also referred to as the* vertices *of K.*

We first extend f to a function $\hat{f} : K \to \mathbb{R}$, by setting $\hat{f}(\sigma) := \max\{f(v) : v \in \sigma\}$, for all $\sigma \in K$. Then, we build the filtration over K induced by f and indexed by $t \in \mathbb{R}$ as

$$K_{f,t} := \hat{f}^{-1}((-\infty, t]).$$

Since $t_1 \leq t_2$ implies $(-\infty, t_1] \subseteq (-\infty, t_2]$, it follows that $K_{f,t_1} \subseteq K_{f,t_2}$.

Moreover, each $K_{f,t}$ is itself a simplicial complex. Indeed, if $\sigma \in K_{f,t}$ and $\tau \subseteq \sigma$, then $\tau \in K_{f,t}$ as we have that

$$\hat{f}(\tau) = \max\{f(v) : v \in \tau\} \leq \max\{f(v) : v \in \sigma\} = \hat{f}(\sigma) \leq t,$$

We note that given a *filtration* of a simplicial complex K whose final term is K itself, one can compute the Euler characteristic at each K_i to get a collection of values finishing with $\chi(K)$. This corresponds to the idea of growing a simplicial complex and tracking how its Euler characteristic changes that was introduced when we provided an intuition for the ECT at the begining of this Subsection. We will now focus on building up a particular type *filtrations* that will be the ones used for the ECT.

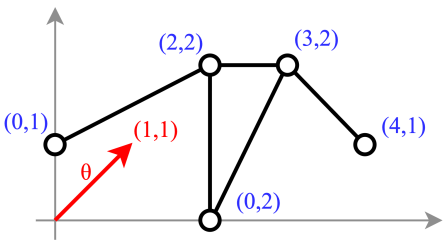
Definition 2.7 *A* featured simplical complex *is a pair (K, x) where K is a simplicial complex and x is a map $x : K^{(0)} \to \mathbb{R}^n$.*

The previous definition extends the concept of a *featured graph* (Definition 2.2), which can be regarded as a 1-dimensional simplicial complex, to simplicial complexes of arbitrary dimension. Hence, every *featured graph* is a particular case of a *featured simplicial complex*. Another example of a *featured simplicial complex* is given by (K, i), where K is a geometric simplicial complex in $\mathbb{R}^n$, and $i : K^{(0)} \subset \mathbb{R}^n \to \mathbb{R}^n$ denotes the identity map restricted to the set of vertices $K^{(0)}$, so that $i(v) = v$ for all $v \in K^{(0)}$.

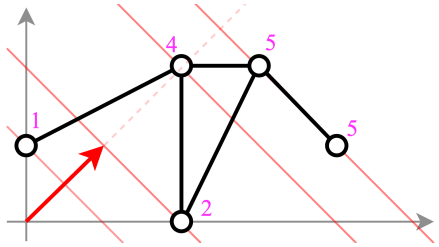
Given a *featured simplical complex* (K, x), one may think of many different ways of building a real valued function from $x : K^{(0)} \to \mathbb{R}^n$ to then induce a filtration over K. We focus on one particular way that will allow us to build a whole family of filtrations over K.

Definition 2.8 *Given a featured simplicial complex (K, x) with $x : K^{(0)} \to \mathbb{R}^n$, and a direction $\theta \in \mathbb{S}^{n-1} \subset \mathbb{R}^n$, we define the* filter function of x through θ, *denoted by x_θ as:*

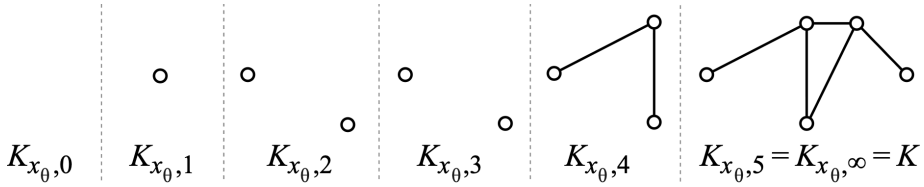
$$\begin{aligned} x_\theta \colon K^{(0)} &\to \mathbb{R} \\ v &\mapsto \langle x(v), \theta \rangle, \end{aligned}$$



(a) Featured simplicial complex (K, x) with a direction $\theta \in \mathbb{S}^1$.



(b) $x_\theta(v)$, $\forall\, v \in K^{(0)}$, where x_θ is the filter function of x through θ.



(c) Filtration over the featured simplicial complex (K, x) induced by the direction θ.

Figure 2.4: Visualization of how direction $\theta \in \mathbb{S}^{d-1}$ induces a filtration for a featured simplicial complex (K, x) with $x : K^{(0)} \to \mathbb{R}$.

where $\langle \cdot, \cdot \rangle$ denotes the standard scalar product in $\mathbb{R}^n$. Moreover, following the notation introduced in Remark 2.6, given $t \in \mathbb{R}$, we use $K_{x_\theta,t}$ to denote $\widehat{x_\theta}^{-1}((-\infty, t])$.

From the definition above we get a way for building filtrations of a featured simplical complex on "every direction", as we described when providing the intuition for the ECT at the begining of the Subsection. In Figure 2.4, we exemplify how a featured simplicial complex together with a direction result in a filtration that captures how the simplicial complex "expands" along the given direction.

We now introduce one more definition that will be the last building block for the ECT.

Definition 2.9 *Given a featured simplical complex (K, x) with $x : K^{(0)} \to \mathbb{R}^n$, and a direction $\theta \in \mathbb{S}^{n-1}$ we denote the* Euler Characteristic Curve of (K, x) in the direction θ *by* $\mathrm{ECC}_\theta[K, x]$*, we define it as follows:*

$$\begin{aligned} \mathrm{ECC}_\theta[K, x] : \mathbb{R} &\to \mathbb{N} \\ t &\mapsto \chi(K_{x_\theta, t}). \end{aligned}$$

In the cases where x is clear, for example when working with a geometric simplical complex, we may simplify the notation and just write $\mathrm{ECC}_\theta[K]$ instead of $\mathrm{ECC}_\theta[K, x]$.

Intuitively speaking, the definition above provides us with a tool to track the Euler characteristic of a featured simplicial complex as it grows along

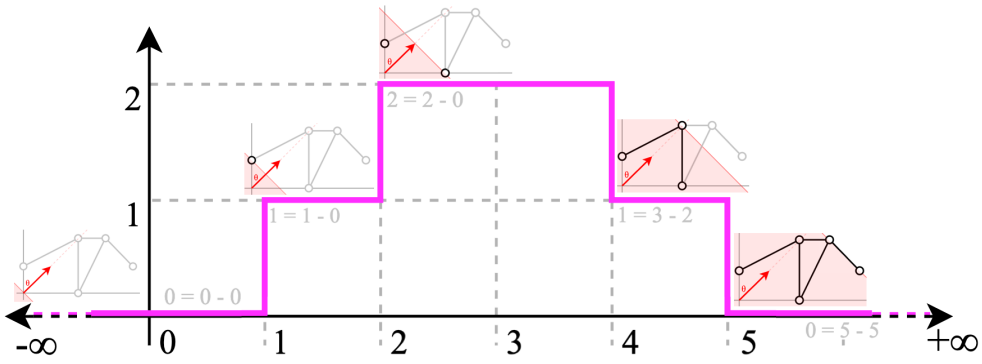


Figure 2.5: Visualization of of $\mathrm{ECC}_\theta[K,x] : \mathbb{R} \to \mathbb{N}$, for the featured simplicial complex (K,x), and the direction θ from Figure 2.4. At each time step (horizontal axis), the ECC takes the value of the Euler characteristic of the corresponding subcomplex in the filtration of (K,x) induced by θ.

a specific direction, we exemplify this in Figure 2.5. For the ECT we will consider all possible directions instead of a fixed one.

Definition 2.10 *Given a featured simplicial complex (K,x) with $x : K^{(0)} \to \mathbb{R}^n$, we define its* Euler Characteristic Transform*, denoted by* $\mathrm{ECT}[K,x]$*, as:*

$$\begin{aligned} \mathrm{ECT}[K,x] : \mathbb{R} \times \mathbb{S}^{n-1} &\to \mathbb{N} \\ (t,\theta) &\mapsto \mathrm{ECC}_\theta[K,x](t) \end{aligned}$$

A very interesting property of the ECT, as we advanced at the beginning of this section, is that in some cases it is an injective mapping, that is, if two "shapes" are different their ECT will also be different. Recall that, as we showed in Figure 2.2 this is not the case when solely looking at the Euler characteristic. More precisely, [38] proved the injectivity of the ECT for geometrical simplicial complexes of dimensions 2 and 3. Therefore, if we have two geometrical simplicial complexes $K_1, K_2 \subset \mathbb{R}^d$ with $d \in \{2,3\}$ we can tell if they are different just by looking at $\mathrm{ECT}[K_1]$ and $\mathrm{ECT}[K_2]$. Moreover, this result has been generalized to geometrical simplicial complexes of arbitrary dimension [13, 7]. It has also been shown that even the ECT restricted to a finite subset of directions, or what is the same a finite collection of ECCs, is enough to get injectivity for geometrical simplical complexes [7, 28].

2.2.2 Discretization of the ECT

In the previous subsection, we introduced the ECT of a featured simplicial complex (K,x) as a map $\mathrm{ECT}[K,x] : \mathbb{R} \times \mathbb{S}^{n-1} \to \mathbb{N}$. While this formulation is mathematically precise, it is not particularly well-suited for machine learning models, where we typically aim to work with fixed-size feature representations of each data sample. For this reason, it is convenient to work

with a fixed size discretized approximation of the ECT. To achieve this, we follow the approach presented in [30] and [28], where the continuous map $\mathrm{ECT}[K, x]$ is approximated by a matrix with integer entries. In the remainder of this subsection, we will construct this discretization step by step.

The first step in constructing the discretized version of $\mathrm{ECT}[K, x]$ is to restrict it to a finite subset of directions. This is motivated by the result mentioned in the previous subsection, which states that, for geometric simplicial complexes, a finite set of directions is sufficient for $\mathrm{ECT}[K, x]$ to be injective. Therefore, instead of working with the $\mathrm{ECT}[K, x]$, we restrict the map to the domain $\mathbb{R} \times \{\theta_1, \ldots, \theta_m\}$ with $m \in \mathbb{N}$, which is a subset of the original domain $\mathbb{R} \times \mathbb{S}^{n-1}$. We note that, it is not clear what should be the set of directions $\Theta = \{\theta_1, \ldots, \theta_m\}$. However, as mentioned in [28] choosing "enough" directions at random is often good enough in practice. Moreover, in the next subsection we will see that, in some cases, the directions can even be "learned".

By going from $\mathrm{ECT}[K, x]$ to $\mathrm{ECT}[K, x]|_{\mathbb{R}\times\Theta}$, we have discretized the second term of the cartesian product that constitutes the original domain of $\mathrm{ECT}[K, x]$. If we can now find a finite set $R = \{t_1, \ldots, t_p\} \subset \mathbb{R}$ that we can use to "replace" $\mathbb{R}$ in the product $\mathbb{R} \times \Theta$, we can restrict the ECT to $\mathrm{ECT}[K, x]|_{R\times\Theta}$. We note that $\mathrm{ECT}[K, x]|_{R\times\Theta}$ is a map whose domain is the cartesian product of two finite sets and its image is a subset of $\mathbb{Z}$. Therefore, $\mathrm{ECT}[K, x]|_{R\times\Theta}$ can represented by a matrix with $|\Theta|$ rows and $|R|$ columns where the entry (i, j) is just the value of $\mathrm{ECT}[K, x]|_{R\times\Theta}(t_j, \theta_i) = \mathrm{ECT}[K, x](t_j, \theta_i) \in \mathbb{Z}$. We focus now on how to build the set R.

For this, we will assume that the featured simplicial complex (K, x) with $x : K^{(0)} \to \mathbb{R}^n$ is inside the unit ball $\mathcal{B}(0, 1)$. By this we mean that for all v in $K^{(0)}$ we have that $||x(v)|| \leq 1$. Note that if this is not the case we can always "normalize" the features defining a new map $x' : K^{(0)} \to \mathcal{B}(0, 1)$ such that $x'(v) := cx(v)$ for all $v \in K^{(0)}$; where $c := \max_{v\in K^{(0)}} ||x(v)||$. This maximum is well defined as $K^{(0)}$ is a finite set. Moreover, this normalization could be done in the same way for the cases where K is a geometrical simplicial complex.

Now, given a featured simplical complex (K, x) with $x : K^{(0)} \to \mathbb{R}^n$ and $||x(v)|| \leq 1$ for all $v \in K^{(0)}$, we note that, for any direction $\theta \in \mathbb{S}^{n-1}$ we have that $x_\theta(v) \in [-1, 1]$ for all $v \in K^{(0)}$. This follows from the definition of x_θ (Definition 2.8) and the Cauchy–Schwarz inequality. Since $x_\theta(v) := \langle x(v), \theta\rangle$ for all $v \in K^{(0)}$, and, by Cauchy-Schwarz we get that

$$|\langle x(v), \theta\rangle| \leq ||x(v)|| \cdot ||\theta|| = 1 \cdot 1 = 1,$$

from where it follows that $x_\theta(v) \in [-1, 1]$ for all $v \in K^{(0)}$.

For this reason, we can restrict even more the domain of $\mathrm{ECT}(K, x)|_{\mathbb{R}\times\Theta}$ and work with $\mathrm{ECT}(K, x)|_{[-1,1]\times\Theta}$. Then, to accomplish our goal of restricting

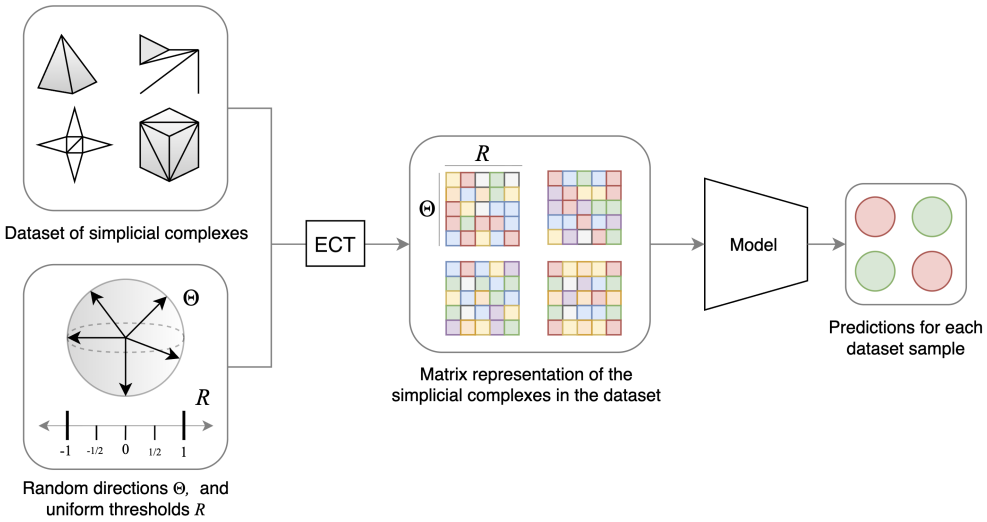


Figure 2.6: Example of how the discretized version of the ECT can be used to generate static feature matrices for simplicial complexes to train a model and make predictions about them.

the original ECT to a finite domain $R \times \Theta$ we can define R as $\{t_0, t_1, \dots t_{q-1}\}$ where $q \in \mathbb{N}$ is a fixed number that decides the number of discretization steps and $t_i = -1 + 2i/(q-1)$. That is, R is a finite set of q elements evenly spaced along the interval $[-1, 1]$.

Figure 2.6 shows how each simplicial complex in a dataset can be mapped to a matrix via the discretized ECT, enabling these matrices to be used as hand-crafted input features to train a model for a prediction task.

Finally, we highlight that this discretization of the ECT is convenient for multiple reasons. First, as we previously mentioned, $\mathrm{ECT}[K, x]|_{R\times\Theta}$ can be represented by a fixed size matrix with $|\Theta|$ rows and $|R|$ columns where the entry (i, j) is just the value of $\mathrm{ECT}[K, x](t_j, \theta_i)$. Second, the "user" can decide how many points in $[0, 1]$ and directions in $\mathbb{S}^{n-1}$ to use to "tune" the discretized approximation of the ECT depending on its compute power, dataset, task complexity etc.

2.2.3 Differentiable ECT

So far, we have introduced the ECT and provided a "recipe" for constructing fixed-size approximations of it, which can be naturally represented as matrices. This enables the use of the ECT as a form of hand-crafted features in machine learning pipelines. In describing this approach, we have worked with a featured simplicial complex (K, x), where the vertex features are defined statically by a function $x : K^{(0)} \to \mathbb{R}^n$.

However, we note that, if the matrix representation of $\mathrm{ECT}(K, x)|_{R\times\Theta}$ could be computed in a way that is differentiable with respect to the feature function

$x(\cdot)$, then $\mathrm{ECT}(K,x)|_{R\times\Theta}$ could be incorporated as a layer in deep learning architectures operating on learnable (i.e., non-static) features. Similarly, if $\mathrm{ECT}(K,x)|_{R\times\Theta}$ was differentiable with respect to the directions $\theta_i \in \Theta$, we could replace the randomly chosen directions with *learned* ones, that is, we could optimize the directions along which the ECT is computed.[5]

[30] provide a differentiable approximation of $\mathrm{ECT}(K,x)|_{R\times\Theta}$ that fulfills the previously described requirements. Moreover, on their work they empirically show how their differentiable approximation of the ECT can be used in end-to-end trainable deep learning architectures. In the remaining of this section we will introduce the differentiable approximation of the ECT by [30].

First, we note that, given a featured simplicial complex (K,x) with $x : K^{(0)} \to \mathbb{R}^n$, and $(t,\theta) \in \mathbb{R}\times\mathbb{S}^{n-1}$, we can rewrite $\mathrm{ECT}[K,x](t,\theta)$ in the following way:

$$\begin{aligned}
\mathrm{ECT}[K,x](t,\theta) &:= \mathrm{ECC}_\theta[K,x](t) && \text{by Def 2.10 (ECT)}\\
&= \chi(K_{x_\theta,t}) && \text{by Def 2.9 (ECC)}\\
&= \sum_{i=0}^{d}(-1)^i |K_{x_\theta,t}{}^{(i)}| && \text{by Def 2.4 } (\chi(\cdot))\\
&= \sum_{i=0}^{d}(-1)^i \sum_{\sigma\in K^{(i)}} \mathbb{1}_{[-\infty,\,\widehat{x}_\theta(\sigma))}(t), && \text{by Def 2.8 } (K_{x_\theta,t})
\end{aligned}$$

where, following the notation introduced in Remark 2.6, $\widehat{x}_\theta(\sigma) = \max_{v\in\sigma} x_\theta(v)$.

The previous rewriting of the ECT in terms of indicator functions is convenient, as one can replace the indicator function by a differentiable approximation, such as the *sigmoid function*. This is exactly what is done in [30] where the differentiable approximation of $\mathrm{ECT}[K,x]$, that we will denote by $\overline{\mathrm{ECT}}[K,x]$, is defined as:

$$\overline{\mathrm{ECT}}[K,x](t,\theta) := \sum_{i=0}^{d}(-1)^i \sum_{\sigma\in K^{(i)}} \mathcal{S}(\lambda(t-\widehat{x}_\theta(\sigma))), \tag{2.3}$$

for all $(t,\theta) \in \mathbb{R}\times\mathbb{S}^{n-1}$. Note that $\mathcal{S} : \mathbb{R}\to[0,1]$ denotes the *sigmoid function* where $\mathcal{S}(t) := 1/(1+\exp(-t))$, $\forall t \in \mathbb{R}$, and λ is a "scale" parameter that regulates the "tightness" of the approximation to the indicator function.

[5] We require only differentiability, not convexity, which aligns with standard practice in deep learning. As long as gradients can be computed, gradient descent (or its variants) can be used to optimize a function, even without guarantees of reaching a global optimum. This is typical in deep learning contexts.

When looking at the expression in Equation 2.3, the only obstacle for differentiability with respect to the feature map x and the direction θ is $\widehat{x_\theta}$, as its definition requires taking a max over $\langle x(v), \theta \rangle$ for the vertices v in a given simplex. However, it is clear that $\langle x(v), \theta \rangle$ is differentiable with respect to both $x(v)$ and θ. On the other hand, $\max(a, b)$ for $a, b \in \mathbb{R}$ is differentiable whenever $a \neq b$ and its gradient is:

$$\nabla \max(a, b) = \begin{cases} (1, 0) & \text{if } a > b, \\ (0, 1) & \text{if } b > a. \end{cases}$$

For this reason the max operator is often used in deep learning workflows[6], as automatic differentiation libraries such as *PyTorch*, only need to evaluate the gradients in specific points, so that either $(1, 0)$ or $(0, 1)$ will be taken in the cases described above, and $(0.5, 0.5)$ is propagated by convention[7] in the cases where $a = b$.

In [30], this issue with respect to the use of max is not explicitly mention. However, we considered that it was worth adding it here, specially for the more rigorous readers.

To conclude this subsection, we highlight that we have just introduced a differentiable way to approximate the entries of the matrix that represents discretized ECT described in the previous subsection. This allows to use the ECT as a specialized layer in deep learning workflows. This is showcased by [30], where, aside of introducing the differentiable approximation of the ECT, they empirically test its effectiveness integrating it on end-to-end trainable deep learning models. They show that the learnable ECT is fast and computationally efficient and provides a performance comparable to that of more complex models across multiple graph (1-dimensional simplicial complexes) and point cloud (0-dimensional simplicial complexes) datasets.

2.2.4 Local ECT

So far, we have shown how the Euler Characteristic Transform (ECT) can be leveraged to generate, or even "learn", features that allow models to make predictions about featured simplicial complexes, such as featured graphs. When introducing graph learning problems (Section 2.1), we distinguished between *node-level* and *graph-level* tasks, depending on whether predictions are to be made about the entire graphs or about individual nodes. However, the ECT, provides a global descriptor of a given "shape", thus limiting its direct applicability to graph-level tasks.

[6] In fact, ReLU, one of the most popular activation functions, is defined as $\text{ReLU}(a) := \max(0, a)$, $\forall a \in \mathbb{R}$.

[7] Note that $(0.5, 0.5)$ is a valid subgradient of the max function in such cases.

To address this limitation, [43] proposes a local variant of the ECT, referred to as the *local Euler Characteristic Transform* (l-ECT), which makes the ECT suitable for node-level tasks in graph learning. In this subsection, we briefly introduce the l-ECT and summarize the contributions of [43].

In the work by [43] the l-ECT is defined for geometric simiplicial complexes. However, to stay consitent with the previous subsections, we will define it for the more general notion of featured simplicial complexes.

Definition 2.11 *Given a featured simplicial complex (K, x) with $x : K^{(0)} \to \mathbb{R}^n$, and a vertex $v \in K^{(0)}$, we define the* local ECT *(l-ECT) of v with respect to $k \in \mathbb{N}$ as*

$$l\text{-ECT}_k[K, x; v] := \text{ECT}[\mathcal{N}_k(v, K), x|_{\mathcal{N}_k(v,K)}], \tag{2.4}$$

where $\mathcal{N}_k(v, K)$ denotes a local neighborhood *of $v \in K^{(0)}$, whose locality scale is controlled by k.*

In the definition by [43], the *local neighborhood* $\mathcal{N}_k(v, K)$ can either refer to the the full subcomplex of K spanned by the the k-hop neighbors of v, or the full subcomplex of K, which is spanned by the *k-nearest vertices* of v. Although [43] refers to the *nearest vertices* in a geoemtric simplicial complex, we can also keep this notion for featured simplicial complexes (K, x) by using $||x(v_\star) - x(\cdot)||$ to get the *nearest vertices* of a vertex $v_\star \in K^{(0)}$.[8]

Given a featured simplicial complex (K, x), the l-ECT of a vertex $v \in K^{(0)}$ is just the ECT of a subcomplex around that vertex. This is a simple, yet powerful idea, as it allows to generate "local" features for each of the nodes in a featured graph. This enables the use of ECT-based features for node-level tasks in graph learning. In fact, aside of introducing the l-ECT, the work by [43] focuses on its application for node-level tasks on featured graphs combining both theoretical insights and empirical testing of the l-ECT.

In particular, they introduce the following result relating the l-ECT to message passing neural networks.

Theorem 2.12 *Let (G, x) be a featured graph, and let $\{l\text{-ECT}_1[K, x; v]\}_v$ be the set of the 1-hop l-ECTs of all the vertices $v \in V(G)$. Then $\{l\text{-ECT}_1[K, x; v]\}_v$ provides all the (non-learnable) needed information to perform a single message passing step on (G, x).*

By the needed non-learnable information for a single message passing step, [43] mean that for any given vertex $v \in V(G)$ one can theoretically recover the features of its neighboring nodes from its l-ECT. The proof of this result follows from the injectivity of the ECT for geometric simplicial complexes, and that the point cloud formed by the vertices in the 1-hop neighborhood

[8] Note that $d(v_1, v_2) = ||x(v_1) - x(v_2)||$ is not an actual distance in $K^{(0)}$ unless we require that $x(v_1) \neq x(v_2)$ for all $v_1, v_2 \in K^0$ such that $v_1 \neq v_2$.

of v can be seen as a 0-dimensional geometric simplicial complex when all the vertices have different features.

This result highlights the power of the 1-hop l-ECT for graph learning. Furthermore, one can also leverage the k-hop l-ECT with higher values of k to generate even richer features for each of the nodes in a graph.

Moreover, in their work [43] also prove that using the ECT one can perform subgraph counting, which is one of the limitations of traditional message passing architectures [6]. Therefore, ECT-based methods can, in some cases, be more powerful than traditional message passing. From this, the authors highlight the potential of designing new hybrid architectures that combine message passing and ECT variants.

When it comes to the experimental part of their work, the authors focus on node-level tasks for featured graphs. They compare l-ECT based approaches to multiple message passing architectures. They show how, for several datasets, their approach that uses the l-ECT either on the 1-hop neighborhoods, the 2-hop neighborhoods, or on features from both hops outperforms traditional graph architectures.

In contrast to [30], they do not use the ECT in an end-to-end trainable fashion in combination with deep learning architectures. Instead, they compute the l-ECT for each node using its discretized version (see Subsection 2.2.2) , and use this as static hand-crafted features to train an XGBoost [5] classifier, rather than using a neural network architecture.

Lastly, we mention that the authors of [43] also propose a Rotation-Invariant Metric based on the l-ECT, and show how it can be used to learn spatial alignment of geometric graphs. However, we will not expand on this here as it lies out of the scope of this work.

Chapter 3

Methods

We propose a new learnable positional encoding for graph learning problems based on the l-ECT. We first introduce the PE, and present a series of remarks that provide insights and motivation for our proposed approach. We then discuss different strategies for projecting the discretized ECT of a given shape into a lower-dimensional space, since such a projection constitutes one of the steps in our approach. Finally, we discuss the limitations of our approach.

3.1 l-ECT based PE

Given a featured graph (G, x), where the node features have dimension d and may be either *static* (i.e. the original node features or some other PE) or *learned* (i. e. the hidden states of the nodes at some step in a MPNN), let $R \subset [0,1]$ denote a set of thresholds and $\Theta \subset \mathbb{S}^{d-1}$ a set of directions. The *LEAP* PE (l-ECT and Projection PE), of a node $v \in V(G)$ is constructed as follows:

1. Compute the m-hop subgraph $\mathcal{N}_m(v, G)$, around the node v.
2. Given the set of nodes $N = \{u_1, \ldots, u_n\} \in V(\mathcal{N}_m(v, G))$, their features $\{x(u_1), \ldots, x(u_n)\}$ are mean centered and then divided by the maximum norm in the set so that they are inside the unit ball. This results in a new set of node features $F = \{f(u_1), \ldots, f(u_n)\} \in \mathbb{S}^{d-1}$, where $f : N \to F$ denotes the map between each node in the m-hop and its normalized feature vector.
3. Compute a matrix $M \in \mathbb{R}^{|\Theta| \times |R|}$ where each entry (i, j) is the differentiable approximation of the ECT of $(\mathcal{N}_m(v, G), f)$ at $(t_j, \theta_i) \in R \times \Theta$. That is, $M_{i,j} = \overline{\text{ECT}}[\mathcal{N}_m(v, G), f](t_j, \theta_i)$. The expression for $\overline{\text{ECT}}$ is provided in Equation 2.3.
4. Lastly, a learnable projection $\varphi : \mathbb{R}^{|\Theta| \times |R|} \to \mathbb{R}^k$ is used to map M to a

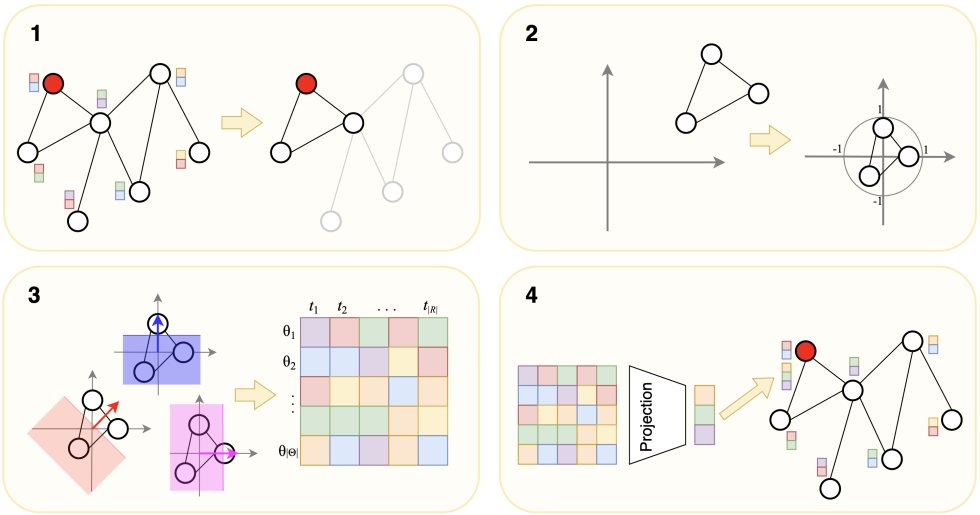


Figure 3.1: Steps of the l-ECT based PE using 1-hop neighborhoods. (1) The neighborhood of a node in a featured graph is selected. (2) Nomalize the neighborhood. (3) Computation of the differentiable ECT. (4) Project the matrix representation of the ECT to generate PE.

vector $\mathrm{PE}(v) \in \mathbb{R}^k$, which is the final positional encoding of the node v. We provide more details about the projection φ in the next subsection.

In Figure 3.1 we illustrate the four steps of our proposed l-ECT PE, we will refer to it as *LEAP PE* or just *LEAP*.

Remark 3.1 LEAP *is not a static pre-processing step on the graph like LaPE or RWPE. On the contrary, our proposed PE can be integrated in a Graph Learning learning architecture to be trained on and end-to-end fashion.*

With the previous remark, we stress the main difference between LEAP and LaPE or RWPE, and also with the way in which the l-ECT is used in [43], where it acts as non-learnable way for extending node features, and the connectivity of the neighborhoods is disregarded as the ECT is computed in the neighborhoods as if they were pointclouds and not graphs.

It should also be noted that our approach can be applied on learned graph features, as $\overline{\mathrm{ECT}}$ is differentiable almost everywhere with respect to the node features in which it is being computed. This means that, when learned graph features are used for LEAP, they can be optimized during training to be "useful" for the $\overline{\mathrm{ECT}}$. We also highlight that, as in [30] the directions in Θ can be randomly chosen and either stay fixed or be be optimized during training. The number of directions in Θ and the number of thresholds in R have to be chosen by the user.

Remark 3.2 *Under the categorization by [27] the LEAP PE would be a* local structural encoding.

The local aspect of LEAP PE comes from the fact the PE of each node is computed taking into account only a subgraph around that node. The "locality" of the PE can be controlled by the user as the number of hops for building the subgraphs is a hyperparameter of the model. As a default choice we suggest using just 1-hops as this value provided good results in our experiments, and also in the experiments by the authors of the original l-ECT paper [43]. Furthermore, we comment on two ways of making the PE "more global":

1. Use m-hops with a higher value of m. This approach is straight forward but one should note that it is possible for two nodes to have a different structure in their m-hop neighborhoods but the same structure if their $(m+1)$-hop neighborhoods are considered. Moreover, for a large enough m all the nodes will have the same PE.

2. A pontentially better suited alternative is to compute LEAP PE multiple times for each node increasing the value of m at each time, then, all of them can be concatenated to get a final l-ECT based PE that tracks how the m-hop neighborhoods of the nodes evolve as m grows.

When it comes to why we consider the LEAP PE *structural* rather than positional, we note that the our proposed PE does not contain information from the position of each node in the graph (or in the subgraph). On the contrary, two nodes with the same m-hop neighborhoods would have the same LEAP PE as the output of the second step in the PE computation would be the same for both nodes. This directly matches de description of *local structural encoding* provided by [27]: *"Given an SE of radius m, the more similar the m-hop subgraphs around two nodes are, the closer their local SE will be"*.

Moreover, given a node in the graph, If the normalized features in its m-hop neighborhood happened to form a valid graph embedding [1], and we could access the "real" ECT of that subgraph rather than an approximation, then, thanks to the injectivity results for the ECT (Subsection 2.2.1), we would have all the information needed to recover the neighborhood's structure. One of our experiments (Subsection 4.2.2), is particularly designed to test the ability of the ECT approximation together with a projection to capture topological features of a graph.

We also note that our proposed PE does not exactly use the l-ECT, since the ECT is not applied directly to the raw m-hop neighborhood of a node but rather to a normalized version of it. This normalization is necessary to guarantee that the features lie within the unit ball, which in what allows to restrict the thresholds to the interval $[-1, 1]$ (see Subsection 2.2.2).[2]

[1] Note that there is no guarantee for this happening.

[2] Alternative normalization schemes that avoid dividing by the maximum could also be considered, and might lead to better gradient flow during training. In this work, however, we

Finally, it is worth emphasizing that our proposed PE is not "pure" in the sense that it depends not only on structural properties of the graph but also on the node features (whether static or learnable). In this regard, it can also be interpreted as an enhanced convolution that integrates both node feature information and structural information.

3.2 ECT projection strategies

In the previous subsection, we explained that LEAP PE is computed by applying a learnable projection to the l-ECT of each node. Since the LEAP aims to capture information about the structure around each node, it should ideally be invariant to the orientation or rotation of the neighborhood features. For this reason, the learned projection should be permutation invariant with respect to the ECCs. However, this requirement is often disregarded in practice when working with the ECT [30].

In what follows, we propose five alternative strategies for projecting the l-ECT into a lower-dimensional representation. Among these, two are not permutation invariant, while the remaining three satisfy this property.

Linear projection In this approach, the l-ECT of each node is first flattened so that its matrix representation becomes a vector $v \in \mathbb{R}^D$, where $D = |\Theta| \cdot |R|$. A linear projection is then applied by multiplying v with a learnable matrix $M \in \mathbb{R}^{k \times D}$, where k is the target dimensionality of the l-ECT-based PE. The entries of M are optimized during training, so the number of learnable parameters with respect to $|\Theta|$ and $|R|$ in this strategy is $\mathcal{O}(|\Theta| \cdot |R|)$.

Note that this projection, is not permutation invariant with respect to the order of the directions. However, permutation invariance could be enforced through parameter tying by constraining each row of M to be the concatenation of $|\Theta|$ identical vectors of size $|R|$.

1-**Dimensional Convolutions** This approach is inspired by [31]. The l-ECT of each node is treated as a temporal series with multiple channels. Therefore, the thresholds $t \in R$ are treated as time steps, and each direction $\theta \in \Theta$ represents a channel of the signal. Several convolutions with learnable weights are concatenated, then the signal is averaged across the final channels resulting in a vector $v \in \mathbb{R}^{|R|}$. Lastly, an MLP is used to project the vector v into the final dimension of the positional encoding. Note that this projection, is not permutation invariant with respect to the order of the directions as reordering the channels of a signal changes the output of the convolutions. The number of learnable parameters with respect to $|\Theta|$ and $|R|$ this strategy is $\mathcal{O}(|\Theta| + |R|)$.

restrict ourselves to the proposed simple normalization for simplicity.

DeepSets In this approach, the l-ECT of each node is treated as set of $|R|$-dimensional vectors, each of them representing an ECC along a given $\theta \in \Theta$. This set is processed using an architecture inspired by DeepSets [47], which is a Deep Learning architecture designed to handle sets as an input. More precisely, a given set of vectors corresponding to ECCs is processed in the following way:

$$\text{DeepSets}(\{\text{ECC}_1, \ldots, \text{ECC}_{|\Theta|}\}) := \text{MLP}_2 \left(\sum_{i=1}^{|\Theta|} \text{MLP}_1(\text{ECC}_i) \right),$$

with $\text{MLP}_1 : \mathbb{R}^{|R|} \to \mathbb{R}^h$, where h is a hyperparameter of the architecture, and $\text{MLP}_2 : \mathbb{R}^h \to \mathbb{R}^k$, where k is the dimension of the PE. By construction, this projection strategy is permutation invariant with respect to the directions of the ECT and its number of learnable parameters is independent of $|\Theta|$.

Attention Our second proposal for a projection strategy that is permutation invariant with respect to the directions of the l-ECT is based on the *attention mechanism* (see Subsection 2.1.2). We treat the l-ECT of each node as a set of $|\Theta|$ vectors in $\mathbb{R}^{|R|}$, corresponding to the ECCs along the different directions. This set is processed by a Transformer encoder [39], which is inherently permutation equivariant. The encoder maps each element $\text{ECC}_i \in \mathbb{R}^{|R|}$ to a hidden representation $\text{T}(\text{ECC}_i) \in \mathbb{R}^d$. So, given the l-ECT of a node, applying the Transformer encoder results in a set

$$S = \{\text{T}(\text{ECC}_1), \ldots, \text{T}(\text{ECC}_{|\Theta|})\} \subset \mathbb{R}^d,$$

where d is the hidden dimension of the Transformer encoder. To obtain the l-ECT-based PE, an $\text{MLP} : \mathbb{R}^d \to \mathbb{R}^k$ is applied to the sum of the elements of S, with k being PE dimension. Note that the number of learnable parameters in this projection strategy depends on $|R|$ but not on $|\Theta|$.

Attention with PE This projection strategy is a variation of the previous one. Specifically, instead of feeding the Transformer encoder the set of ECCs directly, we concatenate each ECC_i with its corresponding direction $\theta_i \in \Theta$ before passing it to the encoder. This results in a projection strategy that is still permutation invariant, while additionally incorporating information about the directions along which the ECCs were computed. As in the previous strategy, the number of learnable parameters of this projection is independent of $|\Theta|$.

3.3 Limitations

While the proposed l-ECT–based positional encoding (LEAP PE) offers a flexible and learnable way to integrate graph structural information into deep

learning architectures, it is not without limitations. We highlight some key considerations below:

Dependence on node features As we already mentioned, the encoding is not "purely structural" : it relies on the normalized node features within a neighborhood. In settings where features are noisy or not meaningful, the effectiveness of the PE may be reduced.

Use of the ECT Our method relies on a differentiable approximation of the discretized ECT, computed on graphs that may not necessarily be geometrical. The resulting ECT is subsequently projected into a lower-dimensional space. As a consequence, the theoretical guarantees of the exact ECT (e.g., injectivity) may not fully carry over to the learned representations.

Normalization choices The proposed normalization scheme ensures that node features lie within the unit ball, but it may not be optimal for gradient flow or for preserving relevant discriminative properties of the raw features. Alternative schemes could lead to better performance.

Hyperparameter sensitivity The proposed PE has several hyperparameters, most notably the scale factor for the differentiable approximation of the ECT, the number of directions $|\Theta|$, and the number of discretization steps $|R|$. It is not a priori clear how to set these values, and different choices may affect both performance and computational cost. Prior work also suggests that there is an interplay between these parameters [31]. However, in our experiments we used the same fixed configuration across architectures and datasets and still observed consistently good performance.

Scalability Computing the l-ECT for each node requires retrieving subgraphs around each node in the graph and computing their ECT, which may become expensive for large graphs or for large values of m. Moreover, since the PE is learned, it cannot be computed once as a preprocessing step, as is the case with LaPE or RWPE. However, in cases where the PE is computed over fixed node features and fixed random directions are used instead of learnable ones, the first three steps of the PE can be carried out as preprocessing, and only the final learnable projection needs to be recomputed at each training step.

Chapter 4

Experiments

4.1 Architectures and Baselines

We use three different architectures across our experiments:

1. *GCN*: as introduced in Subsection 2.1.1, GCN is a well established message passing neural network architecture. The used architecture replicates the one used as baseline in [21] and [30].
2. *GAT*: this is another common type of message passing neural network as we have also introduced in Subsection 2.1.1. The used architecture replicates the one used as baseline in [21] and [30].
3. *NoMP*: to test our proposed PE in settings where the original graph structure is altered or ignored by the architecture, we also experiment with a model that does not perform message passing, which we refer to as *NoMP* (short for *no message passing*). This architecture treats the graph as a set of nodes. First, an MLP is applied at node level, projecting the features of each node into a chosen hidden dimension. These projected node features are then fed into a *Transformer encoder* (see Subsection 2.1.2). The resulting node representations are aggregated and passed through another MLP to produce a single prediction for graph-level tasks. Alternatively, for node-level tasks, the transformed features of each node could be processed separately by an MLP to make node-level predictions.

When assessing the performance of our proposed PE, we consider three different baselines:

1. *No PE*: the node features are fed into the chosen architecture without any additional positional encoding.
2. *RWPE*: the Random Walk Positional Encoding (introduced in Subsection 2.1.3) of each node is concatenated to its corresponding node

features before being passed to the architecture. We recall that, under the categorization by [27], RWPE is *local structural PE*, which is the same type as LEAP, making this a particularly appropriate baseline.

3. *LaPE*: the Laplacian Positional Encoding (introduced in Subsection 2.1.3) of each node is concatenated to its corresponding node features before being passed to the architecture. Although, unlike LEAP, LaPE is a *global positional PE*, we still consider this baseline interesting as LaPE is one of the most popular graph PEs.

4.2 Datasets and tasks

We put LEAP to test on nine real-world different graph datasets, and we do an extra experiment on a synthetic dataset.

4.2.1 Real-world datasets

Seven of the real-world datasets correspond to graph-level classification tasks from the TU Benchmark [23] where the nodes have geometrical features. We summarize them in Table 4.1.

Name	Graphs	Avg. Nodes	Avg. Edges	Dim	Domain	Classes
Letter-High [29]	2250	4.67	4.50	2	CV	15
Letter-Med [29]	2250	4.67	3.21	2	CV	15
Letter-Low [29]	2250	4.68	3.13	2	CV	15
Fingerprint [29]	2149	7.06	5.76	2	CV	15
COX2 [37]	467	41.22	43.45	3	SM	2
BZR [37]	405	35.75	38.36	3	SM	2
DHFR [37]	756	42.43	44.54	3	SM	2

Table 4.1: Dataset name and statistics for 7 graph-level classification tasks taken from the TU benchmark. CV indicates that the dataset belongs to the Computer Vision family, and SM indicates that it belongs to the Small Molecules family. Dim indicates the dimension of the geometrical node features.

The 8-th real-world dataset used for our experiments is *alchemy_full* [4], which is also from the TU Benchmark, particularly from the Small Molecules category. This dataset has 202,579 graphs with 10.10 nodes on average, and 10.44 edges on average. The nodes have 3-dimensional geometrical features. For each graph in the dataset there are 12 graph-level regression targets. The tasks at hand are rotation invariant, that is, they do not depend on rotation and translation of the graph features. When working with this dataset the models will be optimized on the 12 tasks at the same time. We normalize the targets so that they are all in the same scale to prevent some of the tasks from weighting more in the loss function.

The 9-th real-world dataset used for our experiments is the *HIV* dataset from the Molecule Net benchmark [44]. This dataset has 41,127 graphs with 25.5 nodes on average, and 54.9 edges on average. In this case, instead of geometrical features, the nodes have 9 categorical features, each of them with a different number of classes ranging from 2 to 54. The learning problem for this dataset is a binary classification graph-level task.

When working with the *HIV* dataset, before feeding the nodes features into the graph architectures, we embed the discrete node features into a continuous space. This embeddings are learned end-to-end with the rest of the architecture. Since the dimension of the node features is much higher than in the other cases, wen using LEAP we add an MLP that projects the learned continuous node features into $\mathbb{R}^3$. Therefore, in this dataset, instead of the original node features, we use learned node features to compute the l-ECT.

4.2.2 Synthetic dataset

We introduce a synthetic dataset designed to test the ability of our approach to capture structural and topological information from graphs independently of node features.

To construct this dataset, we sample 10,000 sets of three points in $\mathbb{R}^2$ within the unit ball. For each set of three points, we generate four graphs that are added to the dataset: the first with no edges, the second with a single random edge, the third with two random edges, and the fourth with all three possible edges.

The learning task consists of predicting the number of edges (equivalently, the number of connected components) of each graph. Note that this task depends only on the graph structure and it is completely independent of the node features. We compare the performance of a GCN against an architecture where the ECT is first computed on the small graphs, then projected into a low-dimensional representation as in LEAP PE, and finally processed by an MLP classifier.

4.3 Experimental Setup

For every evaluated model–dataset combination, we use a 5-fold cross-validation. In each fold, 80% of the data is used for training, while the remaining 20% is held out as the test set. Within each train/test split, 25% of the training set is set aside as a validation set, which is employed for early stopping and model selection. For all the classification datasets the models are trained to minimize the *cross entropy loss*. For the regression dataset

(*alchemy_full*) the models are trained to minimize the *mean squared error* of the 12 tasks at the same time with normalized targets.

Each model is trained for up to 100 epochs, with training automatically terminated early if the validation performance does not improve for 10 consecutive epochs. For each fold, evaluation is performed on the test set using the model weights that achieved the best validation performance during training. We train the models using the Adam optimizer [17] with a batch size of 16, except for *alchemy_full*, where the batch size is increased to 32 due to the larger number of graphs in this dataset compared to the others.

For the GCN and GAT configurations, we use models with 5 layers and 32-dimensional hidden states (as it is done in [30, 21]) on all datasets except *alchemy_full*. Since the *alchemy_full* dataset is considerably larger than the others and involves 12 regression targets per graph, we increase the model capacity by using 10 layers and 64-dimensional hidden states in both MPNNs. For the NoMP models, we always use a single attention head with a hidden dimension of 16 and a feedforward dimension of 16. This configuration was chosen as it approximately matches the number of trainable parameters in our MPNNs.

Lastly, we use 10-dimensional PEs in all cases. The only difference between a model without PE and the same model with PE is that the initial node state dimension is increased by 10. For the ECT computation, we always use 16 directions and 16 thresholds, and set the smoothing parameter for the differentiable indicator functions (see Equation 2.3) to 128, except in the case of the *HIV* dataset, where it is reduced to 64 in order to obtain a "smoother" approximation. This was done to aim for a better gradient flow as in the *HIV* dataset LEAP PE is applied on learned features. Finally, the configuration of the different projection strategies for LEAP PE is kept fixed across all considered datasets.

Chapter 5

Results and Discussion

In this chapter, we present the results of our experiments comparing the performance of LEAP PE against different baselines, across the various architectures and datasets described in Section 4. Unless stated otherwise, all results for LEAP PE correspond to the linear projection strategy. In the tables and figures, we use *LEAP rd* to denote the l-ECT-based PE with fixed random directions, and *LEAP ld* when the directions are learned during training.

5.1 Message Passing

In Table 5.1 we show the accuracy results of the GCN architecture for the Computer Vision datasets from the TU Benchmark when using our approach and the different baselines. Across all four datasets, LEAP PE, with either random or learned directions, consistently outperforms the baselines. Within the *Letter* datasets, the variant with learned directions achieves slightly higher accuracy than the one with fixed random directions; however, the difference between these two variants is substantially smaller than the performance gap separating either of them from the other baselines. The largest margin is observed in the *Letter-high* dataset, where the LEAP yields a relative accuracy gain of $\sim$80% with respect to the worst-performing baseline (No PE).

In Table 5.2 we show the accuracy results of the GAT architecture for the Computer Vision datasets from the TU Benchmark. The overall trend is consistent with that observed for the GCN results (Table 5.1); across all four datasets, LEAP PE consistently outperforms the baselines. Moreover, when comparing these results to those in Table 5.1, the performance differences between GCN and GAT are smaller than the accuracy gains obtained from incorporating LEAP PE in either architecture.

In Table 5.3, the results for the Small Molecules datasets are presented. In all cases, using LEAP PE with learnable directions yields the best results, beating

Table 5.1: Accuracy results for different PE strategies when using a GCN architecture for multiple Computer Vision datasets from TU Benchmark. Best results are **bold green**, second best are green, and worst are red. For every dataset, our approach achieves the best and second best results.

	Letter-high	Letter-med	Letter-low	Fingerprint
No PE	41.6 ± 4.1	63.5 ± 2.0	80.4 ± 1.0	48.8 ± 1.4
RWPE	60.9 ± 1.7	68.9 ± 2.7	83.2 ± 1.4	49.4 ± 0.6
LaPE	55.3 ± 2.6	75.8 ± 2.6	89.2 ± 1.2	48.1 ± 1.8
LEAP rd	72.2 ± 3.3	82.8 ± 1.4	95.2 ± 0.9	**55.6 ± 1.1**
LEAP ld	**74.2 ± 1.5**	**83.6 ± 1.3**	**96.0 ± 0.9**	54.7 ± 1.5

Table 5.2: Accuracy results for different PE strategies when using a GCN architecture for multiple Computer Vision datasets from TU Benchmark. Best results are **bold green**, second best are green, and worst are red. For every dataset, our approach achieves the best and second best results.

	Letter-high	Letter-med	Letter-low	Fingerprint
No PE	41.9 ± 3.2	58.4 ± 3.7	89.4 ± 0.7	50.5 ± 0.6
RWPE	63.0 ± 3.0	69.0 ± 1.8	90.8 ± 1.5	50.4 ± 0.8
LaPE	54.7 ± 5.3	75.2 ± 2.3	89.6 ± 1.5	48.9 ± 1.0
LEAP rd	70.2 ± 2.2	82.4 ± 2.3	**95.8 ± 0.8**	**55.1 ± 0.6**
LEAP ld	**73.5 ± 2.1**	**82.4 ± 1.6**	95.1 ± 0.9	54.8 ± 2.1

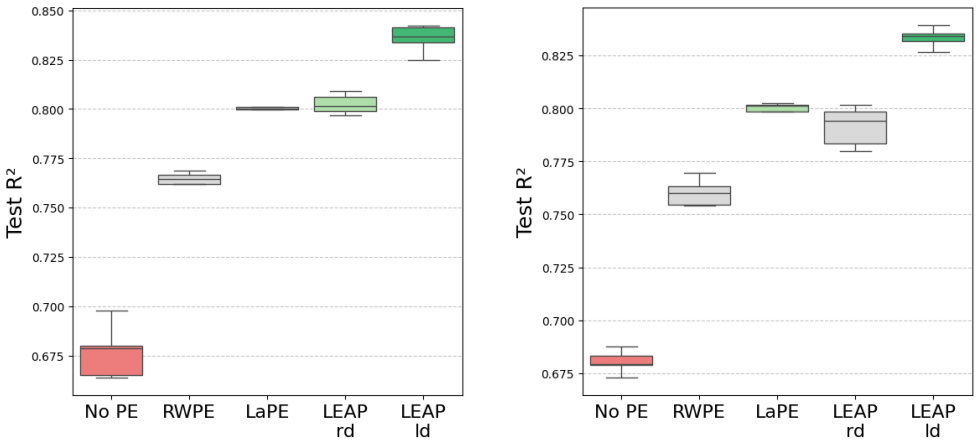
all the baselines. Compared to the Computer Vision datasets (Tables 5.1 and 5.2), the performance differences between the various baselines and our approach are less pronounced. This may be attributed to the considerably smaller size of these datasets making it harder to profit from more complex features. Indeed, in Table 5.3, the dataset exhibiting the largest accuracy gain from the use of LEAP is *DHFR*, which is also the dataset with more graphs among the three, although it is still considerably smaller than the Computer Vision datasets.

Figure 5.1 illustrates the results for the *alchemy_full* dataset. The best performance is clearly obtained when using LEAP with learnable directions. More generally, in the figure we also observe a clear benefit by incorporating positional encodings in both MPNNs, this is aligned with the results in Table 5.1, and Table 5.1 and also with previous work on graph learning [12, 11]. Note that R^2 was reported instead of accuracy because this is a regression task instead of a classification one.

Figure 5.2 presents the results for the *HIV* dataset. The results show a high degree of variability, and this is the only case in which neither of the

Table 5.3: Accuracy results for different PE strategies when using a GCN and GAT architectures for multiple Small Molecules datasets from TU Benchmark. Best results are **bold green**, second best are green, and worst are red. For every dataset-architecture combination, our approach with learnable directions achieves the best result.

	COX2		BZR		DHFR	
	GCN	GAT	GCN	GAT	GCN	GAT
No PE	77.9 ± 1.0	78.2 ± 0.6	81.9 ± 3.3	80.5 ± 2.0	71.6 ± 1.4	73.7 ± 1.8
RWPE	78.4 ± 0.5	79.0 ± 1.4	79.5 ± 2.2	78.3 ± 1.1	73.0 ± 2.4	70.9 ± 2.4
LaPE	78.4 ± 0.9	77.9 ± 1.0	80.3 ± 1.2	80.3 ± 1.2	70.4 ± 2.8	70.4 ± 2.7
LEAP rd	79.2 ± 0.6	78.4 ± 1.2	78.8 ± 0.6	79.3 ± 1.8	75.7 ± 3.0	75.7 ± 3.0
LEAP ld	**79.4 ± 1.0**	**79.7 ± 2.0**	**82.5 ± 2.0**	**82.9 ± 2.9**	**77.8 ± 2.5**	**76.3 ± 2.2**



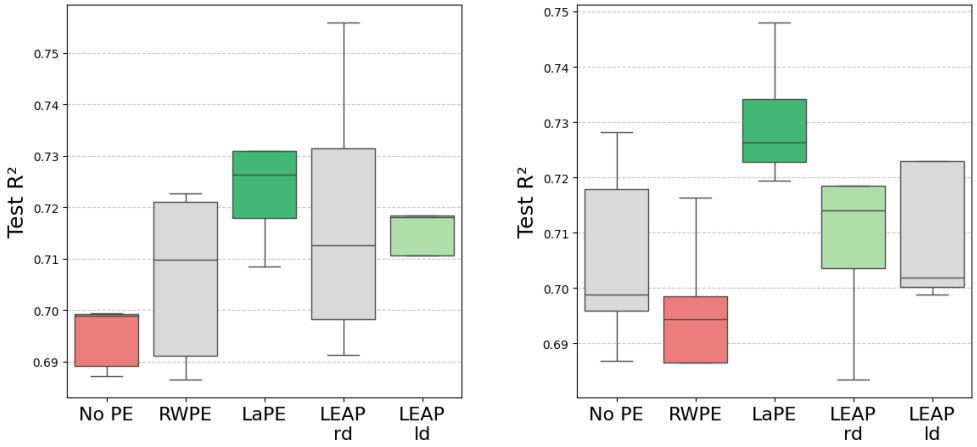
(a) GCN **(b)** GAT

Figure 5.1: R^2 results for different PE strategies on the *alchemy_full* dataset using GCN and GAT architectures. Best results are **bold green**, second best are green, and worst are red. Our approach with learnable directions achieves the best result for both architectures.

two LEAP PE variants outperforms all baselines. Nevertheless, both LEAP variants achieve higher accuracy than RWPE, which belongs to the same PE category, namely *local structural encoding*. Using LaPE results in the strongest performance, suggesting that *global positional* information, which is not captured by the l-ECT, may be particularly relevant in this dataset. Note that AUROC was reported instead of accuracy because this the two categories of the classification task are heavily unbalanced.

5.2 No Message Passing

In Table 5.4 and Table 5.5 we report the accuracy results for the NoMP architecture on the TU Computer Vision and TU Small Molecule datasets, respectively. The overall trend is consistent with the results obtained using



(a) GCN **(b)** GAT

Figure 5.2: AUROC results for different PE strategies on the *HIV* dataset using GCN and GAT architectures. Best results are **bold green**, second best are green, and worst are red. LaPE achieves the best result for both architectures. For both architectures one of the two variants of our approach achieves the second best result

Table 5.4: Accuracy results for different PE strategies when using NoMP architecture for multiple Computer Vision datasets from TU Benchmark. Best results are **bold green**, second best are green, and worst are red. For every dataset, our approach achieves the best and second best results.

	Letter-high	Letter-med	Letter-low	Fingerprint
No PE	63.4 ± 1.0	57.8 ± 0.9	89.7 ± 1.3	50.7 ± 0.5
RWPE	79.2 ± 1.4	84.5 ± 1.3	94.7 ± 1.0	51.3 ± 0.7
LaPE	65.0 ± 1.6	76.9 ± 2.5	91.1 ± 2.2	50.5 ± 1.2
LEAP rd	**79.5 ± 1.2**	**86.0 ± 2.2**	**96.7 ± 0.8**	55.7 ± 1.1
LEAP ld	79.4 ± 1.1	85.4 ± 1.5	96.4 ± 0.7	**56.3 ± 1.4**

GCN and GAT (Tables 5.1, 5.2, and 5.3): in all cases, LEAP PE, with either random or learnable directions, outperforms the baselines.

An interesting observation arises for datasets such as *Letter-high* and *Letter-low*, where using NoMP without any PE yields better performance than MPNNs without PEs. This highlights the limitations of MPNNs, and how, in some cases, even models without any information about the graph structure can yield better results than these architectures.

5.3 Evaluation of ECT projection methods

To evaluate the different projection strategies that we proposed for LEAP (see Section 3.2), we repeat all experiments on the classification datasets from

Table 5.5: Accuracy results for different PE strategies when using NoMP architecture for multiple Small Molecules datasets from TU Benchmark. Best results are **bold green**, second best are green, and worst are red. For every dataset, our approach achieves the best and second best results.

	COX2	BZR	DHFR
No PE	77.9 ± 0.8	79.8 ± 2.6	70.1 ± 3.4
RWPE	77.7 ± 1.3	80.9 ± 1.7	73.3 ± 1.5
LaPE	77.7 ± 1.0	81.2 + 3.2	70.5 ± 3.5
LEAP rd	**79.0 ± 0.6**	83.2 ± 1.7	74.3 ± 6.1
LEAP ld	78.6 ± 0.8	**84.7 ± 2.7**	**74.9 ± 3.3**

Table 5.6: Best approach (architecture, PE strategy, and projection strategy) and relative accuracy improvement with respect to the worst performing baseline for TU classification datasets. In all cases the best result was achieved using our PE strategy.

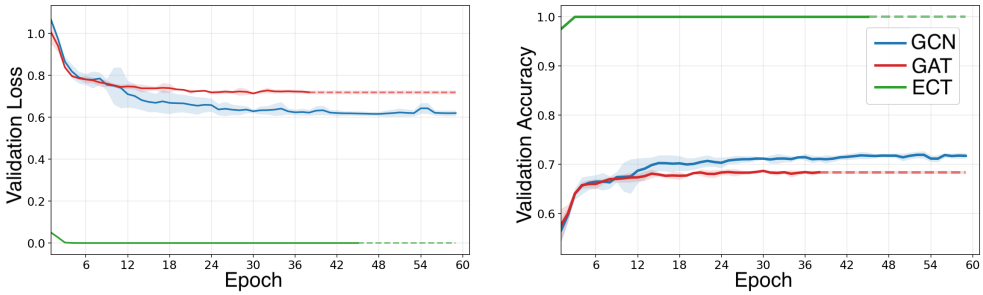
Dataset	Best approach			% Improv.
Letter-high	NoMP	LEAP ld	1D Conv	96.2
Letter-med	NoMP	LEAP ld	1D Conv	53.1
Letter-low	NoMP	LEAP ld	1D Conv	21.9
Fingerprint	NoMP	LEAP ld	Linear	14.1
COX2	GAT	LEAP ld	Attn w/ PE	3.1
BZR	NoMP	LEAP ld	Linear	8.2
DHFR	GCN	LEAP ld	Attn w/ PE	10.7

the TU benchmark using the five proposed projection strategies, each with both fixed and learned directions. The complete tables with these results are provided in Appendix A. In most cases, no projection strategy clearly outperformed the others. In the few situations where one did (e.g., GCN or GAT for *Letter-high*), the linear projection yielded the best results.

As a summary of the results for different projection strategies, Table 5.6 reports the best combination of architecture, PE and projection strategy (if applicable) for each TU classification dataset.

5.4 Synthetic dataset

Figure 5.3 shows the validation loss and accuracy on the *Synthetic* dataset for GCN, GAT, and the ECT+MLP model. The ECT-based model converges rapidly to 100% accuracy and near-zero loss, demonstrating its ability to capture structural properties, which are independent of node features, such as the number of connected components (the classification target in this task). In contrast, GCN and GAT converge more slowly and plateau below 80%



(a) Validation Loss (lower is better) **(b)** Validation Accuracy (higher is better)

Figure 5.3: Validation metrics per training epoch for the synthetic dataset for GCN, GAT, and our ECT-based architecture. Our method achieves a perfect score in both metrics and also converges faster. The shadows around the curves display the standard deviation over 5 runs. The dashed line means that training of the model finished earlier because of early stopping.

accuracy, underscoring the limitations of these architectures.

Test accuracies were 71.83 ± 0.27 for GCN, 69.44 ± 0.82 for GAT, and 100.0 ± 0.0 for ECT, consistent with both validation and training results, confirming that the weaker performance of GCN and GAT is not due to overfitting.

Chapter 6

Conclusions and Future Work

In this thesis, we have presented *LEAP* a new *local structural positional encoding* for graph learning based on the local Euler Characteristic Transform (l-ECT). To the best of our knowledge, this is the first approach that integrates the l-ECT into deep learning architectures in an end-to-end trainable fashion. This allows both the directions of the transform and its projection to be optimized jointly with the learning task.

Our experimental results show that the proposed PE achieves consistently strong results across multiple architectures and datasets beating well established baselines such as Laplacian Positional Encoding (LaPE) and Random Walk Positional Encoding (RWPE). Moreover, our evaluation also shows that learning the directions of the transform improves model performance in most of the evaluated tasks, highlighting the benefits of making this step trainable.

In addition to real-world datasets, we also designed a synthetic dataset aimed at evaluating the ability of our approach to capture structural properties of the graphs independently of node features. On this task, our approach achieved perfect accuracy, clearly demonstrating its ability to capture topological information that the evaluated message passing architectures (GCN/GAT) alone failed to recover.

Taken together, these results underline the potential of l-ECT-based encodings as a promising component in graph learning pipelines. In particular, they are well suited for architectures that rely on global attention mechanisms, where the graph structure is not directly taken into account by the model and multiple positional encodings are often combined to capture complementary aspects of graph structure. We expect that LEAP can provide valuable local structural information in such settings.

6.1 Future Work

We highlight several future research directions related to our work that we identify as particularly interesting:

Broader evaluation Extending our experiments to additional datasets and tasks, particularly in settings where LEAP is applied to learnable features, could provide valuable insights. Different PEs capture complementary aspects of graph structure. Thus, combining LEAP with other PEs or embedding it within more sophisticated architectures may further improve performance.

Threshold optimization Rather than fixing the thresholds of the discretized ECT uniformly, they could be treated as trainable model parameters. The differentiable ECT (Equation 2.3) is differentiable with respect to the thresholds in the same way that it with respect to the directions.

ECT as pooling and convolution Since the l-ECT PE provides a fixed size representation for variable size neighborhoods in a featured graph, future work could explore using the ECT as a pooling operator. Another direction is to design l-ECT-based convolutional layers, where the ECT serves as the aggregation step in message passing neural networks (see Equation 2.1.1)

Gradient flow An interesting direction is to study how the hyperparameters of LEAP influence training dynamics. In particular, it would be useful to investigate how the gradient flow is affected by the scale parameter of the differentiable ECT and by the normalization strategy for m-hop neighborhoods in the l-ECT based PE.

Appendix A

Additional Results

In this appendix we provide additional results that explore the performance of our proposed PE in different settings and when different projection strategies are used.

In Table A.1 we display the accuracy results for the TU Computer Vision datasets for GCN and GAT architectures making different use of the PEs. In Table A.2 we show the results for the same experiments but for the Small Molecules datasets. In particular:

- *PE name, no features*: the graph neural network only received the PE of each node as features, without the original node features.
- *PE name, learnable PE*: follows the approach of using a learned PE at each message passing steps described by [12]. The indicated PE is used as additional node features before the first message passing step. From there onward, the PE of each node is updated at each message passing step taking into account the current PE and the PE of the neighbors.
- *LEAP + RWPE*: the architecture uses as PE the concatenation of our l-ECT based PE with RWPE.
- *LEAP at each step*: our proposed PE is computed on the node features at each message passing step and appended to the node features before applying the next step.

In Table A.3 and Table A.4 we display the accuracy results for the experiments with different projection strategies for LEAP PE (see Section 3.2). These are the results omitted in Section 5.3 in the Results chapter (Chapter 5).

In Figure A.1 and Figure A.2, we show the validation accuracy on the *Letter-high* and *Letter-low* datasets for our proposed PE and the baselines (No PE, RWPE, and LaPE). For both datasets, using LEAP not only yields higher accuracy but also accelerates model convergence. This behavior is consistent

Table A.1: Complementary accuracy results for GCN and GAT for TU Computer Vision datasets exploring multiple ways of integrating LEAP PE.

	Letter-High		Letter-Med		Letter-Low		Fingerprint	
	GCN	GAT	GCN	GAT	GCN	GAT	GCN	GAT
RWPE, no features	31.0 ± 2.3	31.3 ± 3.8	33.2 ± 2.4	33.3 ± 2.4	37.3 ± 0.9	37.4 ± 1.5	26.8 ± 1.4	27.5 ± 1.0
LEAP rd, no features	68.2 ± 1.4	68.8 ± 1.2	78.8 ± 1.0	76.8 ± 1.4	93.1 ± 1.1	93.2 ± 1.5	50.6 ± 1.9	50.5 ± 0.9
LEAP rd, learnable PE	73.5 ± 1.4	73.2 ± 1.1	84.0 ± 1.4	84.5 ± 2.3	95.7 ± 1.2	96.4 ± 0.8	55.7 ± 1.6	55.1 ± 55.1
LEAP rd + RWPE	75.1 ± 1.5	72.7 ± 1.5	84.3 ± 2.1	83.8 ± 2.1	96.1 ± 0.9	95.3 ± 0.8	56.0 ± 0.7	55.0 ± 0.4
LEAP rd at each step	71.1 ± 1.8	70.7 ± 2.6	83.9 ± 0.6	84.4 ± 0.8	95.7 ± 0.9	95.6 ± 1.2	54.1 ± 1.2	54.6 ± 1.3
LEAP ld at each step	70.0 ± 3.1	70.7 ± 1.6	83.2 ± 1.8	82.8 ± 1.6	95.7 ± 0.8	95.6 ± 0.9	54.5 ± 1.7	53.8 ± 1.4

Table A.2: Complementary accuracy results for GCN and GAT for TU Small Molecules datasets exploring multiple ways of integrating the l-ECT PE.

	COX2		BZR		DHFR	
	GCN	GAT	GCN	GAT	GCN	GAT
RWPE, no features	78.2 ± 0.5	78.2 ± 0.5	78.8 ± 0.6	78.8 ± 0.6	60.9 ± 0.1	60.9 ± 0.1
LEAP rd, no features	78.2 ± 0.5	78.2 ± 0.6	78.8 ± 0.6	80.2 ± 2.3	60.9 ± 0.1	75.3 ± 1.7
LEAP rd, learnable PE	78.4 ± 1.6	78.2 ± 0.9	81.2 ± 2.7	82.9 ± 2.9	66.7 ± 3.2	76.1 ± 1.7
LEAP rd + RWPE	78.6 ± 0.8	79.0 ± 1.1	81.7 ± 2.5	81.2 ± 1.4	77.9 ± 3.7	77.8 ± 2.5
LEAP rd at each step	79.2 ± 1.4	78.8 ± 0.9	79.8 ± 2.4	81.7 ± 2.4	75.0 ± 2.5	75.5 ± 3.8
LEAP ld at each step	78.2 ± 0.5	78.2 ± 1.4	81.5 ± 2.6	80.7 ± 2.7	75.7 ± 3.1	76.3 ± 3.1

Table A.3: Accuracy results for multiple TU Computer Vision datasets when using different projection strategies of the l-ECT based PE with fixed and learnable directions for different models.

Model	Proj.	Letter-high		Letter-med		Letter-low		Fingerprint	
	Method	fixed dir	learn dir	fixed dir	learn dir	fixed dir	learn dir	fixed dir	learn dir
GCN	Linear	72.2 ± 3.3	74.2 ± 1.5	82.8 ± 1.4	83.6 ± 1.3	95.2 ± 0.9	96.0 ± 0.9	55.6 ± 1.1	54.7 ± 1.5
	Attn	59.8 ± 2.8	62.7 ± 2.6	74.4 ± 1.5	73.9 ± 4.6	92.3 ± 1.3	94.5 ± 1.4	53.0 ± 1.9	54.4 ± 1.2
	Attn PE	67.2 ± 1.5	68.6 ± 1.8	82.0 ± 0.8	82.9 ± 2.2	95.8 ± 1.1	94.7 ± 1.6	54.1 ± 1.3	55.1 ± 1.2
	DeepSets	59.2 ± 1.9	63.4 ± 2.1	72.4 ± 0.6	76.0 ± 1.1	91.4 ± 1.8	92.0 ± 0.6	52.3 ± 1.3	54.1 ± 1.3
	1D Conv	66.4 ± 2.8	63.1 ± 3.2	81.6 ± 1.5	81.7 ± 3.5	94.2 ± 1.5	93.4 ± 1.4	55.6 ± 1.1	54.0 ± 2.2
GAT	Linear	70.2 ± 2.2	73.5 ± 2.1	82.4 ± 2.3	82.4 ± 1.6	95.8 ± 0.8	95.1 ± 0.9	55.1 ± 0.6	54.8 ± 2.1
	Attn	62.0 ± 1.2	62.9 ± 3.1	75.0 ± 2.1	79.7 ± 1.4	94.4 ± 0.1	93.4 ± 1.0	52.0 ± 1.3	53.5 ± 1.5
	Attn PE	66.7 ± 2.0	65.0 ± 3.5	83.2 ± 1.1	79.3 ± 2.4	94.0 ± 1.3	95.1 ± 1.1	54.5 ± 1.8	54.5 ± 1.8
	DeepSets	58.8 ± 2.6	56.4 ± 4.3	75.4 ± 3.1	76.6 ± 2.3	94.6 ± 1.4	93.1 ± 1.3	52.9 ± 2.2	51.0 ± 1.1
	1D Conv	67.3 ± 2.0	68.1 ± 1.3	80.9 ± 1.3	80.5 ± 2.8	93.6 ± 2.0	95.2 ± 0.9	54.8 ± 1.5	54.9 ± 0.7
NoMP	Linear	79.5 ± 1.2	79.4 ± 1.1	86.0 ± 2.2	85.4 ± 1.5	96.7 ± 0.8	96.4 ± 0.7	55.7 ± 1.1	56.3 ± 1.4
	Attn	79.0 ± 1.78	81.3 ± 1.9	84.8 ± 0.9	86.5 ± 2.6	96.1 ± 0.6	97.2 ± 0.8	53.8 ± 0.7	54.8 ± 1.2
	Attn PE	81.3 ± 1.9	80.6 ± 3.6	88.0 ± 1.9	86.5 ± 2.2	96.3 ± 0.8	96.2 ± 1.2	54.8 ± 1.4	55.3 ± 1.1
	DeepSets	78.0 ± 2.0	79.2 ± 1.9	86.0 ± 2.2	87.5 ± 2.0	96.3 ± 0.6	96.6 ± 0.8	54.0 ± 0.4	54.3 ± 1.0
	1D Conv	81.1 ± 0.9	81.6 ± 1.9	87.0 ± 2.0	88.5 ± 2.5	97.2 ± 0.3	98.0 ± 0.4	54.5 ± 1.0	54.1 ± 0.4

with what we observed on the synthetic dataset (see Section 5.4) and was also observed during training on other datasets.

Table A.4: Accuracy results for multiple TU Small Molecules datasets when using different projection strategies of LEAP PE with fixed and learnable directions for different models.

Model	Proj. Method	COX2		BZR		DHFR	
		fixed dir	learn dir	fixed dir	learn dir	fixed dir	learn dir
GCN	Linear	79.2 ± 0.6	79.4 ± 1.0	78.8 ± 0.6	82.5 ± 2.0	70.9 ± 3.2	74.9 ± 4.0
	Attn	78.4 ± 1.3	79.0 ± 0.9	81.7 ± 2.8	82.5 ± 1.6	74.1 ± 5.2	77.3 ± 4.1
	Attn PE	78.2 ± 1.2	78.8 ± 1.3	82.5 ± 2.4	82.5 ± 3.1	73.2 ± 3.7	77.6 ± 2.8
	DeepSets	78.2 ± 0.6	79.0 ± 1.2	79.0 ± 1.2	81.7 ± 3.5	71.2 ± 2.6	73.3 ± 3.6
	1D Conv	78.0 ± 1.2	79.2 ± 2.0	79.8 ± 1.9	82.5 ± 2.7	71.7 ± 1.2	76.7 ± 3.4
GAT	Linear	78.4 ± 1.2	79.7 ± 2.0	79.3 ± 1.8	81.7 ± 2.4	75.7 ± 3.0	76.3 ± 2.2
	Attn	78.4 ± 0.5	78.8 ± 0.8	82.0 ± 3.4	82.2 ± 2.1	74.9 ± 3.1	73.3 ± 2.8
	Attn PE	78.8 ± 0.8	80.1 ± 2.2	82.0 ± 3.2	79.5 ± 0.7	73.2 ± 3.1	75.9 ± 5.8
	DeepSets	79.2 ± 1.6	79.7 ± 1.8	81.2 ± 2.0	82.7 ± 4.7	71.2 ± 4.1	76.5 ± 3.8
	1D Conv	78.4 ± 0.9	78.6 ± 1.7	81.7 ± 4.0	83.7 ± 2.9	70.6 ± 2.3	75.7 ± 1.5
NoMP	Linear	79.0 ± 0.6	78.6 ± 0.8	83.2 ± 1.7	84.7 ± 2.7	74.3 ± 6.1	74.9 ± 3.3
	Attn	78.2 ± 0.5	78.2 ± 0.5	81.7 ± 2.8	79.0 ± 0.9	68.3 ± 5.7	71.7 ± 4.1
	Attn PE	78.4 ± 0.4	77.7 ± 1.5	78.8 ± 0.6	78.8 ± 0.6	64.2 ± 4.8	72.1 ± 3.7
	DeepSets	78.4 ± 0.8	78.0 ± 0.5	83.2 ± 2.1	82.0 ± 2.8	69.5 ± 3.0	72.6 ± 3.6
	1D Conv	78.0 ± 1.9	78.1 ± 1.1	81.0 ± 1.7	79.3 ± 1.8	71.6 ± 3.2	75.7 ± 2.7

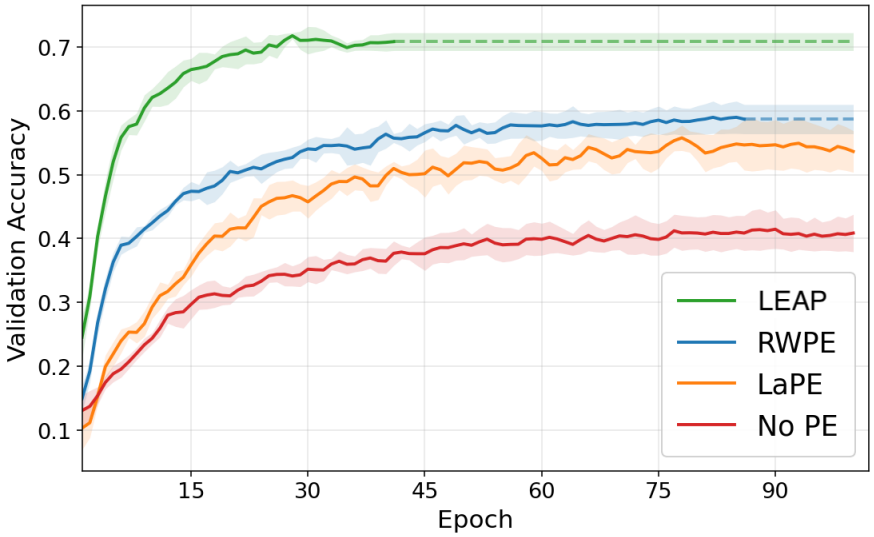


Figure A.1: Validation accuracy per training epoch the *Letter-high* dataset for different PE strategies using a GCN architecture. Our method achieves the best results The shadows around the curves display the standard deviation over 5 runs. The dashed line means that training of the model finished earlier because of early stopping.

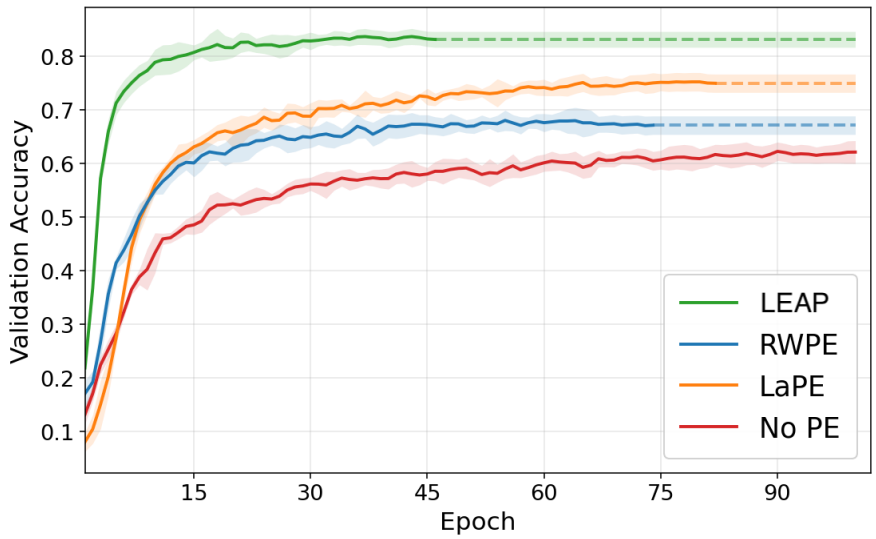


Figure A.2: Validation accuracy per training epoch the *Letter-med* dataset for different PE strategies using a GCN architecture. Our method achieves the best results The shadows around the curves display the standard deviation over 5 runs. The dashed line means that training of the model finished earlier because of early stopping.

Appendix B

Qualitative insights

B.1 Insights on NoMP vs MPNN

As we mentioned in Subsection 5.2, for the *Letter-high* and *Letter-low* datasets, using NoMP without any PE yielded better performance than using the MPNNs without PEs. This is an interesting result as the NoMP model without PE does not have any information about the graph structure.

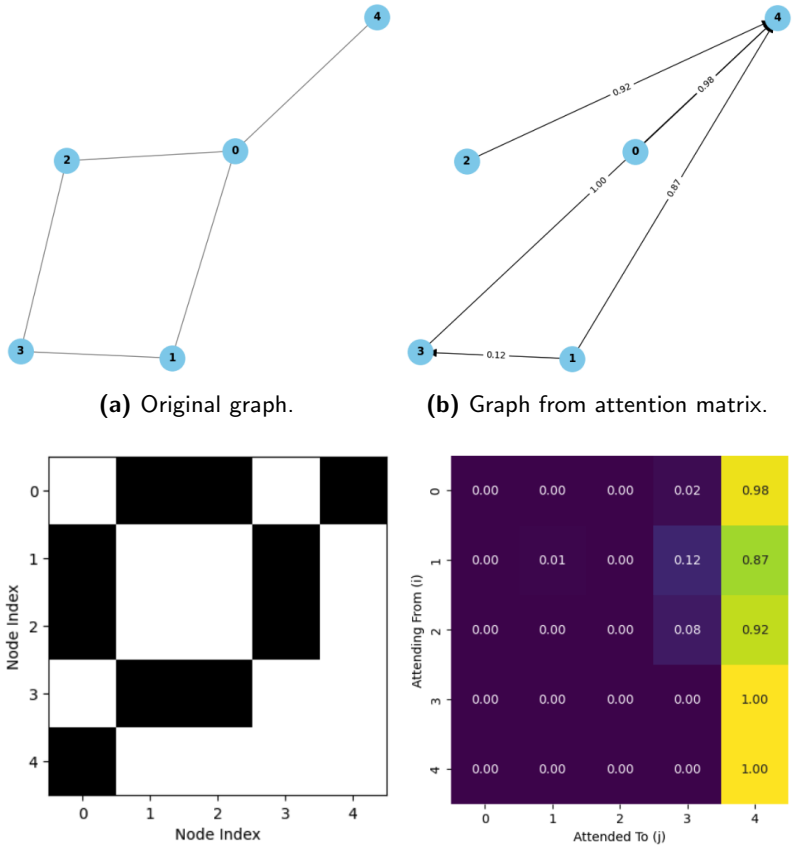
Despite this, we observe that taking into account the graph structure is beneficial for the tasks at hand, as the best overall results on these datasets are obtained when some PE is used. This finding highlights the shortcomings of MPNNs: although they exploit the graph structure, they impose other limitations that, in certain cases, can damage performance.

To gain a better understanding of what is happening in this particular case, we visualized some of the graphs from the dataset together with the *attention coefficients* between nodes learned by the NoMP model. An example of this is shown in Figure B.1. What we see there is that, according to the NoMP model, the information from node 4 is very relevant for every other node. However, when using a MPNN only node 0 has direct access to the information in node 4 as it is its only neighbor. This type of behavior was a common pattern for the graphs that we visualized.

B.2 Visualizing ECT directions

Our results showed that, in general, learning the directions of the ECT is beneficial. To gain some insights on this process we visualized the learned directions for some of the evaluated tasks where node features where 2-dimensional.

In Figure B.2 we show the initial random directions used for the synthetic dataset, as well as the learned directions obtained for different values of



(a) Original graph. **(b)** Graph from attention matrix.

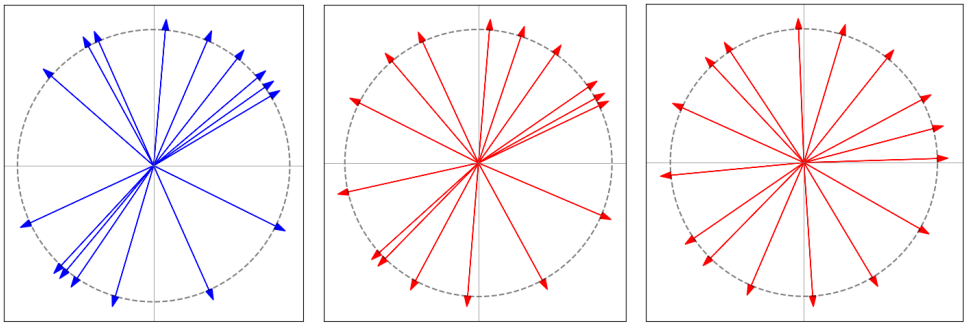
(c) Adjacency matrix **(d)** Attention matrix

Figure B.1: Visualization of a graph randomly sampled from the *Letter-high* dataset and its adjacency matrix. We also show the learned attention coefficients after training the *NoMP* model and the graph that results by keeping the original nodes and adding edges when attention between two nodes is above 0.01.

the smoothing parameter in the differentiable ECT (see Equation 2.3). We observe that, in general, each direction deviates less from its corresponding initial one when the differentiable ECT is "less smooth" (i.e., when the scale parameter is higher). This trend was observed also for the other datasets where we conducted this visualization.

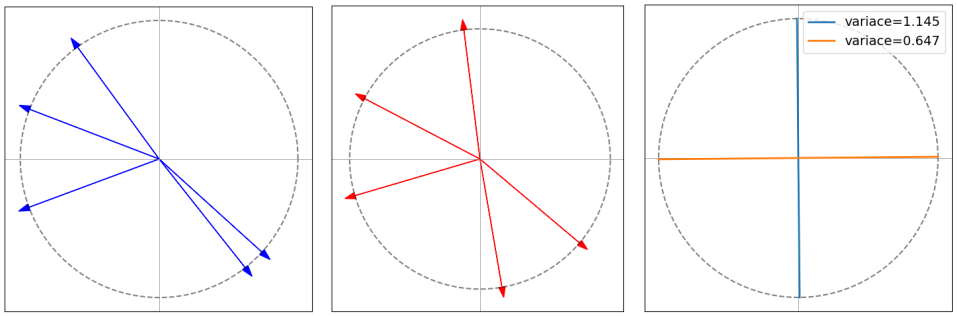
Moreover, by the way in which the synthetic dataset was built, all possible directions in the space should, a priori, be equally relevant. This intuition is confirmed when the smoothing parameter is set to 2, as the learned directions tend to spread more evenly "trying" cover the circumference uniformly.

In Figure B.3 we show the initial random directions used when training a GCN with l-ECT based PE on the *Letter-high* dataset. In this case we use a smaller number of directions. We observed that, when doing this, some of the directions may deviate more from their original position even



(a) Original random directions. **(b)** Learned with scale $= 128$. **(c)** Learned with scale $= 2$.

Figure B.2: Visualization of the ECT directions for the model trained on our synthetic dataset.



(a) Original random directions. **(b)** Learned with scale $= 32$. **(c)** Principal components

Figure B.3: Visualization of the ECT directions for a model trained on Letter-high dataset. In subfigure (c) we show the two principal components of the node features and their variance.

if the scale is not so low. This may be happening, as the model tries to capture an "important" direction in the dataset for which none of the original random ones is close enough. For example, in Subfigure B.3b we see that two directions experimented a significant change and became much more aligned with the vertical axis. If we check the main PCA directions of the node features and their associated variance (Subfigure B.3c), we also see that the PCA direction with highest variance is also almost vertical.

Bibliography

[1] Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between MPNN and graph transformer. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 3408–3430. PMLR, 23–29 Jul 2023.

[2] Semih Cantürk, Renming Liu, Olivier Lapointe-Gagné, Vincent Létourneau, Guy Wolf, Dominique Beaini, and Ladislav Rampášek. Graph positional and structural encoder. *arXiv preprint arXiv:2307.07107*, 2023.

[3] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International conference on machine learning*, pages 3469–3489. PMLR, 2022.

[4] Guangyong Chen, Pengfei Chen, Chang-Yu Hsieh, Chee-Kong Lee, Benben Liao, Renjie Liao, Weiwen Liu, Jiezhong Qiu, Qiming Sun, Jie Tang, et al. Alchemy: A quantum chemistry dataset for benchmarking ai models. *arXiv preprint arXiv:1906.09427*, 2019.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[6] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020.

[7] Justin Curry, Sayan Mukherjee, and Katharine Turner. How many directions determine a shape and other sufficiency results for two topological

transforms. *Transactions of the American Mathematical Society, Series B*, 9(32):1006–1043, 2022.

[8] Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio, and Michael M Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *International conference on machine learning*, pages 7865–7885. PMLR, 2023.

[9] Manfredo Perdigao Do Carmo and J Flaherty Francis. *Riemannian geometry*, volume 2. Springer, 1992.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[11] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

[12] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

[13] Robert Ghrist, Rachel Levanger, and Huy Mai. Persistent homology and euler integral transforms. *Journal of Applied and Computational Topology*, 2(1):55–60, 2018.

[14] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[15] Florian Grötschla, Jiaqing Xie, and Roger Wattenhofer. Benchmarking positional encodings for gnns and graph transformers. *arXiv preprint arXiv:2411.12732*, 2024.

[16] Max Horn, Edward De Brouwer, Michael Moor, Yves Moreau, Bastian Rieck, and Karsten Borgwardt. Topological graph neural networks. In *International Conference on Learning Representations*, 2022.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] TN Kipf. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[19] Chuang Liu, Yibing Zhan, Jia Wu, Chang Li, Bo Du, Wenbin Hu, Tongliang Liu, and Dacheng Tao. Graph pooling for graph neural networks: Progress, challenges, and opportunities. *arXiv preprint arXiv:2204.07321*, 2022.

[20] Liheng Ma, Reihaneh Rabbany, and Adriana Romero-Soriano. Graph attention networks with positional embeddings. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 514–527. Springer, 2021.

[21] Kelly Maggs, Celia Hacker, and Bastian Rieck. Simplicial representation learning with neural k-forms. In *International Conference on Learning Representations*, 2024.

[22] Sohir Maskey, Ali Parviz, Maximilian Thiessen, Hannes Stärk, Ylli Sadikaj, and Haggai Maron. Generalized laplacian positional encoding for graph representation learning. *arXiv preprint arXiv:2210.15956*, 2022.

[23] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[24] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[25] Elizabeth Munch. An invitation to the euler characteristic transform. *The American Mathematical Monthly*, 132(1):15–25, 2025.

[26] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[27] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

[28] Bastian Rieck. Topology meets machine learning: An introduction using the euler characteristic transform. *Notices of the American Mathematical Society*, 72(7):719–727, 2025.

[29] Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pages 287–297. Springer, 2008.

[30] Ernst Röell and Bastian Rieck. Differentiable euler characteristic transforms for shape classification. In *International Conference on Learning Representations*, 2024.

[31] Ernst Röell and Bastian Rieck. Point cloud synthesis using inner product transforms. 2024.

[32] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[33] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

[34] Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, pages 31613–31632. PMLR, 2023.

[35] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

[36] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[37] Jeffrey J Sutherland, Lee A O'brien, and Donald F Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences*, 43(6):1906–1915, 2003.

[38] Katharine Turner, Sayan Mukherjee, and Doug M Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 2014.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[40] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

[41] Yogesh Verma, Amauri H Souza, and Vikas Garg. Topological neural networks go persistent, equivariant, and continuous. *arXiv preprint arXiv:2406.03164*, 2024.

[42] Yogesh Verma, Amauri H Souza, and Vikas Garg. Positional encoding meets persistent homology on graphs. *arXiv preprint arXiv:2506.05814*, 2025.

[43] Julius von Rohrscheidt and Bastian Rieck. Diss-l-ECT: Dissecting graph data with local Euler characteristic transforms. In *Proceedings of the 42nd International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2025.

[44] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[45] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[46] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[47] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[48] Xu Zhang, Yonghui Xu, Wei He, Wei Guo, and Lizhen Cui. A comprehensive review of the oversmoothing in graph neural networks. In *CCF Conference on Computer Supported Cooperative Work and Social Computing*, pages 451–465. Springer, 2023.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. In consultation with the supervisor, one of the following three options must be selected:

○ I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].

○ I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used and cited generative artificial intelligence technologies[2].

◉ I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used generative artificial intelligence technologies[3]. In consultation with the supervisor, I did not cite them.

Title of paper or thesis:

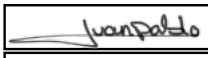
ECT-based Positional Encoding for Graph Neural Networks

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	**First name(s):**
Garcia Amboage	Juan Pablo

With my signature I confirm the following:
- I have adhered to the rules set out in the Citation Guide.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	**Signature(s)**
Zürich, 15/09/2025	

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] E.g. ChatGPT, DALL E 2, Google Bard
[2] E.g. ChatGPT, DALL E 2, Google Bard
[3] E.g. ChatGPT, DALL E 2, Google Bard