Variational Autoencoders for Simplicial Complexes

Modelling Higher Order Relations

Ariella Smofsky

School of Computer Science
McGill University
Montreal, Quebec, Canada

July 2021

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

Abstract

A graph is a combinatorial object that captures a binary relation as a set of edges between pairs of vertices. The generalization of a graph is a hypergraph, consisting of hyperedges between any subset of vertices. A simplicial complex is a hypergraph with additional structure and constraints in which the family of edges is downward closed. In this thesis, we design and implement an end-to-end trainable variational autoencoder for simplicial complexes. This is achieved through the construction of a new representation, encoder, and decoder for simplicial complexes. Our work bridges software libraries for topological data analysis and graph representation learning. We train and evaluate our variational autoencoder for simplicial complexes on the task of relation prediction for higher order relations, the higher order counterpart of link prediction. Finally, we discuss future work at the intersection of topological data analysis and graph representation learning.

Abrégé

Un graphe est un objet combinatoire qui capture une relation binaire par un ensemble d'arêtes entre des paires de sommets. La généralisation d'un graphe est un hypergraphe, constitué d'hyperbords entre tout sous-ensemble de sommets. Un complexe simplicial est un hypergraphe avec une structure et des contraintes supplémentaires dans lequel la famille d'arêtes est fermée vers le bas. Dans cette thèse, nous concevons et implémentons un auto-codeur variationnel entraînable de bout en bout pour les complexes simpliciaux. Ceci est réalisé par la construction d'une nouvelle représentation, d'un encodeur et d'un décodeur pour les complexes simpliciaux. Notre travail établit un lien entre les bibliothèques logicielles pour l'analyse topologiques des données et l'apprentissage de représentations de graphes. Nous formons et évaluons notre auto-codeur variationnel pour les complexes simpliciaux sur la tâche de prédiction de relation pour les relations d'ordre supérieur, la contrepartie d'ordre supérieur de la prédiction de lien. Enfin, nous discutons des travaux futurs à l'intersection de l'analyse topologiques des données et de l'apprentissage de représentations de graphes.

Contributions

This thesis is the product of joint work between my supervisor Prakash Panangaden and myself. Our contributions are:

1. Design and implementation of a new simplicial complex representation, bridging the gap between the topological data analysis library Gudhi [20] and the graph representation learning library PyTorch Geometric [7].

2. Creation of 3 synthetic abstract simplicial complex datasets, including the implementation of code to prepare datasets for training and testing.

3. Design and implementation of two new simplicial neural message passing encoders.

4. Design and implementation of two new decoders for triangle prediction.

5. Definition and implementation of loss functions based on our new simplicial encoders and decoders.

Acknowledgements

I would like to thank everyone who has helped encourage me during my graduate studies. To my supervisor Prakash Panangaden, thank you for your dedication and guidance. I will be forever grateful of your nurturing my sense of adventure in research. To Will Hamilton, thank you for being an exceptional teacher and researcher in all things pertaining to graph representation learning. To my loved ones, thank you for the endless giggles and for reminding me of who I am outside of my work.

Contents

List of Figures

List of Tables

1

Introduction

Link prediction is a popular task within machine learning and is used in a multitude of applications such as drug reactivity [31] and knowledge base completion [22]. Given two entities, the task of link prediction is the inference of whether an edge exists. In this thesis, we define relation prediction as the generalization of link prediction to higher order relations, which are relations between more than two entities. Applications of relation prediction include co-authorship citation networks, online thread participation and drug networks [3]. In particular, this thesis focuses on the task of predicting whether a triangle exists given a triplet of corresponding edges.

This work presents a novel simplicial autoencoder and pipeline; an initial attempt at bridging the gap between topological data analysis and graph representation learning. First, we define a new abstract simplicial complex representation based on the simplex tree [4] and the coordinate list representation of the machine learning library PyTorch Geometric [7]. Next, we present our novel simplicial neural message passing encoders. These encoders are inspired by the highly popular variational graph autoencoder [14]. In order to adapt

the simplicial model's reconstructed output to the relation prediction setting, in particular to the task of triangle prediction, we present two novel decoders. The first is based on multi-relational graph representation learning, the second is based on the Čech complex. Finally, we demonstrate our simplicial autoencoder effectively performs triangle prediction on synthetic abstract simplicial complex datasets.

2

Background

In this chapter we provide the reader with the necessary mathematics and machine learning background to understand our construction of variational autoencoders for simplicial complexes. To start we introduce the simplicial complex as a structurally constrained hypergraph, the generalization of graphs to higher-order relations. These structural constraints are imposed and arise from geometrical considerations. We then give a brief introduction to graph neural networks and graph convolutional networks. We proceed with Kipf et al.'s [14] latent variable machine learning model, variational graph autoencoders. Finally, we review two important concepts in topological data analysis: Vietoris-Rips and Čech complexes, as mechanisms to construct simplicial complexes from points in a metric space.

2.1 Simplicial Complexes

Simplicial complexes generalize graphs from binary relations to higher-order relations. In their simplest form, graphs are pairwise relationships between entities, often referred to as

nodes or vertices. In this thesis, we will use the terms nodes and vertices interchangeably. Formally, graphs can be defined as follows.

Definition 2.1.1 (Graph)**.** *A* directed graph G *is a tuple* (V, E) *consisting of*

- *A set of vertices* V*,*
- *A set* E *of ordered pairs of elements of* V *called edges;* $E \subseteq V \times V$*.*

An undirected graph *is a graph such that* E *is a set of 2-element subsets of vertices* V*.*

As can be seen from the definition, the edges of a graph represent a binary relation between the vertices. We now build toward the definition of a simplicial complex. The building block of a simplicial complex is a simplex, which can have multiple forms.

2.1.1 Geometric Simplicial Complex

Let us start with a preliminary geometry definition.

Definition 2.1.2 (Convex hull)**.** *The* convex hull *of a subset of* k *points* $V = \{v_1, ..., v_k\} \subseteq \mathbb{R}^n$ *is the set of all convex combinations,*

$$\alpha_1 v_1 + \alpha_2 v_2 + ... + \alpha_k v_k \textit{ such that}$$
$$\forall i \in [k].\alpha_i \geqslant 0 \textit{ and } \sum_{i=1}^{k} \alpha_i = 1,$$

of points in set V*.*

Note the convex hull is the smallest convex set containing all the points in set V.

Definition 2.1.3 (Geometric k-simplex in $\mathbb{R}^n$)**.** *A* geometric k-simplex *is the convex hull of* $k+1$ *vertices* $\{v_0, v_1, ..., v_k\} \subseteq \mathbb{R}^n$ *such that* $\{v_1 - v_0, v_2 - v_0, ..., v_k - v_0\}$ *are linearly independent.*

A geometric 1-simplex in $\mathbb{R}^n$ can be viewed as a graph edge incident to vertices v_0 and v_1, embedded in $\mathbb{R}^n$. Naturally graph edges must be embedded in a vector space of at least dimension 1. More generally, a geometric k-simplex embedded in $\mathbb{R}^n$ must have n greater or equal to k. For this reason, a geometric k-simplex is sometimes called a geometric k-dimensional simplex. Similar to unit vectors, there is a standard (unit) geometric k-simplex.

Definition 2.1.4 (Standard geometric k-simplex)**.** *The convex hull of vertices* $\left\{ v \in \mathbb{R}^{k+1} \;\middle|\; \sum_{i=1}^{k+1} v_i = \mathbf{1}, v_i \geqslant 0 \right\}$ *is the* standard geometric k-simplex Δ^k*, where* v_i *is the* i*-th component of vector* v *and* $\mathbf{1}$ *is a vector in* $\mathbb{R}^{k+1}$.

A crucial feature of simplices is their hierarchical definition. The concept of faces allows us to formalize this hierarchy.

Definition 2.1.5 (Face and coface of a geometric k-simplex in $\mathbb{R}^n$)**.** *Let* $L = \{ v_0, v_1, ..., v_k \}$ *be a geometric* k*-simplex. A* d-face *of simplex* L *is the convex hull of a non-empty subset* $D \subseteq L$ *of* $d+1$ *vertices. A* f-coface *of simplex* L *is the convex hull of a superset* $F \supseteq L$ *of* $f+1$ *vertices.*

Note that the face and the coface of a geometric simplex, are themselves geometric simplices. We are now ready to define a geometric simplicial complex.

Definition 2.1.6 (Geometric simplicial complex K in $\mathbb{R}^n$)**.** *A* geometric simplicial complex K *is a set of geometric simplices, of possibly various dimensions, in* $\mathbb{R}^n$ *such that*

- *Every face of each geometric simplex in K is in turn a geometric simplex of K,*
- *The intersection of any two geometric simplices of K is a face of each simplex.*

The dimension of geometric simplicial complex K *is equal to the largest dimension of its constituent geometric simplices.*

The first condition ensures the closure of simplices under subsets. The second condition constrains how simplices can be connected to form a valid geometric simplicial complex.

2.1.2 Abstract Simplicial Complex

An abstract simplicial complex is the purely combinatorial counterpart to a geometric simplicial complex. Abstract simplicial complexes are composed of simplices, which are subsets of vertices rather than geometric convex hulls of these subsets.

Definition 2.1.7 (Abstract simplicial complex K)**.** *Let $V = \{ v_0, v_1, ..., v_N \}$ be a set of vertices. Then* abstract simplicial complex K *consists of subsets of these vertices such that for each $k \geqslant 0$:*

- *K_k is a set of k-*simplices*, subsets of size $k+1$ of vertices V,*
- *Every $(j+1)$-element subset of K_k is an element of K_j , where $j < k$.*

Here, each K_k is an abstract simplicial complex consisting of a set of k-simplices, where each simplex is closed under subsets. Similar to definition 2.1.6, the dimension of abstract simplicial complex K is equal to that of its simplex with the largest dimension. Let us illustrate with an example in Figure 2.1.

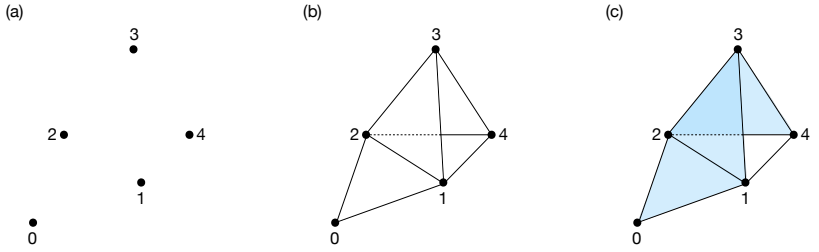


Figure 2.1: An example of a 2-dimensional simplicial complex K where (a) is the set of vertices K_0, (b) is the set of vertices and edges $K_0 \cup K_1$, and (c) is the set of vertices, edges, and triangles $K = K_0 \cup K_1 \cup K_2$.

Definition 2.1.8 (Face and coface of d-simplex σ)**.** *Let $K = K_0 \cup K_1 \cup ... \cup K_k$ be a k-dimensional abstract simplicial complex and $\sigma \in K_d$ be a d-simplex of K. A f-face of simplex σ is a f-simplex belonging to K whose vertices form a subset of σ. A j-coface of simplex σ is a j-simplex belonging to K whose vertices form a superset of σ.*

Note that an abstract simplicial complex of dimension 1 is an undirected graph. Abstract simplicial complexes are ideal for representing binary, ternary, and generally any higher-order undirected relation between entities. The property of closure under subsets ensures a higher-dimensional simplex can only belong to the abstract simplicial complex if all of its lower-dimensional simplices are included. In other words, higher-order relations can only be present if they are built upon the necessary lower-order relations. It is in this manner that tetrahedra are composed of triangles which are composed of edges formed by vertices. Thus, the notion of hierarchical relations is inherent to abstract simplicial complexes.

2.2 Neural Networks for Graphs

A feedforward neural network, also called a multilayer perceptron (MLP), is a compositional model originally proposed by Greenblatt [25] to learn a mathematical function capable of mapping input values to a target output value. In the 2010s, Bengio, LeCun et al. [2][19] trained neural networks with deep architectures to successfully perform tasks such as object detection and speech recognition. A currently active area of research is focused on leveraging the success of neural networks in continuous domains, to discrete domains; in particular, to model relationships between entities. In this section we introduce an important approach to deep learning for relational data: neural networks for graphs.

2.2.1 Feedforward Neural Networks

In the simplest case, feedforward neural networks are used for the task of supervised classification. The aim of supervised classification is to learn a function $f_{\boldsymbol{\theta}} : \mathbb{R}^n \rightarrow [k]$, where k denotes the number of classes the function must learn to map a given input to. The training dataset for the classification task is a set of training examples of the form $(\boldsymbol{x}, y) \in \mathbb{R}^n \times [k]$ where $\boldsymbol{x}$ is the input representation vector and y is the target class.

The key innovation of feedforward neural networks in the deep learning setting is the stacking of layers to automate the extraction of features from input data. These layers $\{ 1, ..., N \}$ are a sequence of linear functions $\{ f_1, f_2, ..., f_N \}$ each followed by a non-linear activation σ, composed together such that the function learned by the neural network

is as follows:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = (f_N \circ \sigma \circ f_{N-1} \circ \sigma \circ ... \circ \sigma \circ f_1)(\boldsymbol{x}). \tag{2.1}$$

The non-linear activation σ is commonly chosen to be the rectilinear unit function $\text{ReLU}(\boldsymbol{h}) = \max(\boldsymbol{0}, \boldsymbol{h})$. Note the function f is parameterized by the set of parameters $\boldsymbol{\theta}$. These parameters are the learnable weights $\boldsymbol{W}_i$ at each layer i:

$$f_i(\boldsymbol{h}) = \boldsymbol{W}_i^\top \boldsymbol{h} + \boldsymbol{b}_i, \tag{2.2}$$

where $\boldsymbol{b}_i$ is the bias term. Training a neural network refers to learning the parameters $\boldsymbol{\theta} = \{ \boldsymbol{W}_1, \boldsymbol{W}_2, ..., \boldsymbol{W}_N \}$ by minimizing a differentiable loss function $\mathcal{L}$ with respect to $\boldsymbol{\theta}$. The loss function $\mathcal{L}$ is the objective function which is optimized using backpropogation [28] in conjunction with stochastic gradient descent. The benefits of the compositionality of neural networks are twofold: first, complex features, represented by deep layers, can be learned from simpler features and the corresponding shallow layers; and secondly, the architecture is amenable to automatic differentiation.

2.2.2 Graph Representation

As seen in section 2.2.1, feed forward neural networks take as input $\boldsymbol{x} \in \mathbb{R}^n$ where n is the number of features in the input representation. This setup works well for tasks with a natural vector representation of the input data. For graph representation learning tasks, two components must be captured: the structure of the graph and the features of the graph.

The structure of a graph refers to its set of edges E, which represent the binary relations

between vertices V. The structure of a graph with n nodes is commonly represented as follows.

Definition 2.2.1 (Adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|V|} \times \mathbb{R}^{|V|}$)**.** *An* adjacency matrix *of graph* $G = (V, E)$ *is a square matrix* $\boldsymbol{A} \in \mathbb{R}^{|V|} \times \mathbb{R}^{|V|}$ *such that:*

- $\boldsymbol{A}[i, j] = 1$ *if* $(i, j) \in E$,
- $\boldsymbol{A}[i, j] = 0$ *if* $(i, j) \notin E$.

A weighted adjacency matrix *is a variant where the entries of matrix* $\boldsymbol{A}$ *are real-valued.*

Here, $\boldsymbol{A}[i, j]$ is notation for the matrix entry $\boldsymbol{A}_{ij}$ and the adjacency matrix of an undirected graph is symmetric. Note the ordering of vertices along the rows and columns of adjacency matrix $\boldsymbol{A}$ is crucial as it alters the matrix. The problem of finding whether two adjacency matrices represent equivalent graphs is known as the graph isomorphism problem and is NP-intermediate. Thus, a vertex order is often assumed. An alternate graph structure representation, convenient for sparse graphs with few edges, is:

Definition 2.2.2 (Coordinate list (COO) of graph G)**.** *The* coordinate list (COO) *of graph* $G = (V, E)$ *is the enumeration of the set* $\{ (i, j) \mid (i, j) \in E \}$.

The features of a graph refer to additional information, aside from connectivity structure, known about a graph. A common type of feature is node-level features, represented by real-valued matrix $\boldsymbol{X} \in \mathbb{R}^{|V| \times D}$, where D is the number of features of each node. Note that entry $\boldsymbol{X}[i] \in \mathbb{R}^{D}$ is the feature vector of node i. For this reason, the node ordering across node features $\boldsymbol{X} \in \mathbb{R}^{|V| \times D}$ and adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|V| \times |V|}$ must be consistent. As we will see in section 2.2.3, graph neural networks require node features. For tasks with no apparent node features, a one-hot indicator feature is used to identify each node.

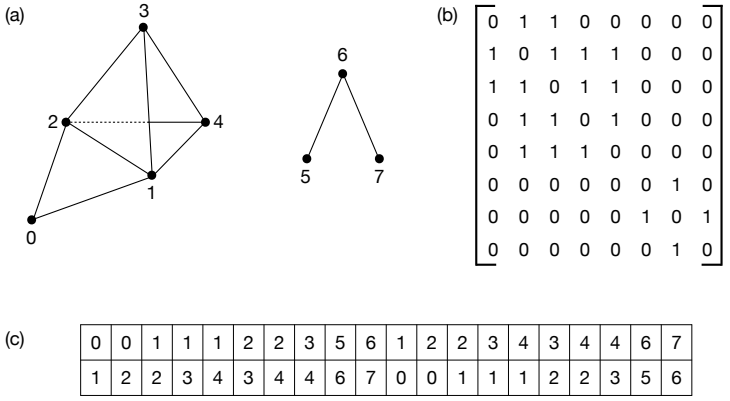


Figure 2.2: (a) is a graph $G = (V, E)$ with $|V| = 8$ and $|E| = 10$. The corresponding representations of graph G are: (b) the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|V|\times|V|}$, and (c) the coordinate list $\boldsymbol{E} \in \mathbb{R}^{2\times 2|E|}$.

Definition 2.2.3 (Node-level one-hot features $\boldsymbol{X} \in \mathbb{R}^{|V|\times|V|}$)**.** *For graph* $G = (V, E)$*,* node-level one-hot features *are a square matrix* $\boldsymbol{X} \in \mathbb{R}^{|V|\times|V|}$ *such that:*

- $\boldsymbol{X}[i, i] = 1$ *for every node* $i \in V$*,*
- $\boldsymbol{X}[i, j] = 0$ *for every node* $i, j \in V, i \neq j$*.*

The level of granularity of features can be adjusted to edges, for which there exists analogous edge-level one-hot features.

2.2.3 Graph Neural Network

Graph neural networks (GNNs) are a specific type of neural network designed for learning functions which take graphs as input. As seen in the discussion about the representation of graphs 2.2.2, graph-structured data is composed of two components: structure and fea-

tures. Graph neural networks essentially combine these two components with a mechanism referred to as *message passing*. In its simplest form, message passing is the process of updating node embeddings according to the local structure of each node. The local structure of a node is formally defined as:

Definition 2.2.4 (Neighbourhood of node u, $\mathcal{N}(u)$)**.** *Given the graph $G = (V, E)$, the* neighbourhood of node $u \in V$*, denoted $\mathcal{N}(u)$, is the set of nodes* $\{ v \mid (u, v) \in E \}$.

Let us consider the graph $G = (V, E)$, represented by adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|V| \times |V|}$, and the node features $\boldsymbol{X} \in \mathbb{R}^{|V| \times D}$. Message passing for node $u \in V$ at iteration $k + 1$ is as follows:

$$\boldsymbol{h}_u^{(k+1)} = \text{UPDATE}\left(\boldsymbol{h}_u^{(k)}, \text{AGGREGATE}\left(\left\{ \boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u) \right\}\right)\right) \tag{2.3}$$

$$= \text{UPDATE}\left(\boldsymbol{h}_u^{(k)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}\right), \tag{2.4}$$

where $\boldsymbol{h}_u^{(k)} \in \mathbb{R}^{D^{(k)}}$ is the node embedding of node u and $\boldsymbol{m}_{\mathcal{N}(u)}^{(k)} \in \mathbb{R}^{D^{(k)}}$ is the message vector aggregated from the node embeddings of neighborhood $\mathcal{N}(u)$. Here, the functions AGGREGATE and UPDATE are differentiable and each node embedding is initialized to the corresponding input node features $\boldsymbol{h}_u^{(0)} = \boldsymbol{X}[v] \in \mathbb{R}^{D^{(0)}}$ where $D^{(0)} = D$. Note the first iteration of message passing can be understood as each node $u \in V$ obtaining information from its one-hop neighbourhood. After k iterations, each node $u \in V$ obtains information from its k-hop neighbourhood, which is the set of nodes that can be reached within k steps of node u.

The foundational GNN model designed by Scarselli et al.[26] selected the following

AGGREGATE and UPDATE functions:

$$\boldsymbol{m}_{\mathcal{N}(u)}^{(k)} = \text{AGGREGATE}\left(\left\{ \boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u) \right\}\right) = \sum_{v \in \mathcal{N}(u)} \boldsymbol{h}_v^{(k)} \tag{2.5a}$$

$$\boldsymbol{h}_u^{(k+1)} = \text{UPDATE}\left(\boldsymbol{h}_u^{(k)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}\right) = \sigma\left(\boldsymbol{W}_{\text{self}}^{(k)} \boldsymbol{h}_u^{(k)} + \boldsymbol{W}_{\text{neigh}}^{(k)} \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}\right). \tag{2.5b}$$

Note the use of a linear function followed by a non-linear activation is reminiscent of the multilayer perceptron in section 2.2.1.

The message passing framework can be generalized to include edge features $\boldsymbol{Y} \in \mathbb{R}^{|E| \times J}$, where J is the number of features of each edge. Message passing for edge $(u, v) \in E$ and node $u \in V$ at iteration $k+1$ is:

$$\boldsymbol{h}_{(u,v)}^{(k+1)} = \text{UPDATE}_e\left(\boldsymbol{h}_{(u,v)}^{(k)}, \boldsymbol{h}_u^{(k)}, \boldsymbol{h}_v^{(k)}\right) \tag{2.6}$$

$$\boldsymbol{m}_{\mathcal{N}(u)}^{(k+1)} = \text{AGGREGATE}\left(\left\{ \boldsymbol{h}_{(v,u)}^{(k+1)}, \forall v \in \mathcal{N}(u) \right\}\right) \tag{2.7}$$

$$\boldsymbol{h}_u^{(k+1)} = \text{UPDATE}_n\left(\boldsymbol{h}_u^{(k)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k+1)}\right), \tag{2.8}$$

where each edge embedding is initialized to $\boldsymbol{h}_{(u,v)}^{(0)} = \boldsymbol{Y}[(u, v)] \in \mathbb{R}^J$. This framework is named the message passing neural network (MPNN), and was designed by Gilmer et al. [11] to unify existing forms of message passing. At each iteration the MPNN updates each edge embedding with its incident node embeddings, aggregates incident edge embeddings to generate a message vector per node, and finally updates each node embedding. The MPNN effectively incorporates both node and edge features into the message passing framework to generate embeddings for the various components of a graph.

The MPNN also generates a global embedding for the entire graph, however this thesis

does not utilize graph-level embeddings. For more details on variants of the GNN, Hamilton's "Graph Representation Learning" [12] is an excellent reference.

2.2.4 Graph Convolutional Network

Graph convolutional networks (GCNs) are a specific type of graph neural network (GNN) designed to be the graph-structured equivalent to image-based convolutional neural networks (CNNs). Authored by Kipf et al. [15], GCNs build upon graph convolutions and message passing GNNs to construct a spectral graph theory motivated model. This section provides a brief discussion of graph convolutions and their incorporation into the graph neural network, to form the graph convolutional network.

Definition 2.2.5 (Continuous convolution $f \star h$)**.** *Let f and h be two real-valued functions on $\mathbb{R}^d$. Then the* continuous convolution of functions f, h *is defined as*

$$(f \star h)(\boldsymbol{x}) = \int_{\mathbb{R}^d} f(\boldsymbol{y}) h(\boldsymbol{x} - \boldsymbol{y}) d\boldsymbol{y}.$$

The connection between convolutions and Fourier transforms from the field of signal processing is particularly useful:

$$(f \star h)(\boldsymbol{x}) = \mathcal{F}^{-1}\left((\mathcal{F}(f(\boldsymbol{x})) \odot \mathcal{F}(h(\boldsymbol{x}))\right), \tag{2.9}$$

where $\mathcal{F}$ is a Fourier transform, $\mathcal{F}^{-1}$ is the inverse Fourier transform, and $\odot$ is the element-wise (Hadamard) product.

Let us consider a graph $G = (V, E)$ represented by adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|V| \times |V|}$, and node features $\boldsymbol{X} \in \mathbb{R}^{|V| \times D}$. We now introduce some foundational spectral theory defini-

tions.

Definition 2.2.6 (Eigenvector v, eigenvalue λ)**.** *Given a linear map $f : \mathbb{R}^n \to \mathbb{R}^n$, a non-zero vector $\boldsymbol{v} \in \mathbb{R}$ is called an* eigenvector *if there exists $\lambda \in \mathbb{R}$ such that $f(\boldsymbol{v}) = \lambda \boldsymbol{v}$. Such a λ is called an* eigenvalue*.*

Definition 2.2.7 (Graph Laplacian operator $\boldsymbol{L}$)**.** *The* graph Laplacian operator $\boldsymbol{L} \in \mathbb{R}^{|V| \times |V|}$ *is defined as:*

6. $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ (unnormalized Laplacian)
7. $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}}$ (normalized Laplacian),

where $\boldsymbol{D} \in \mathbb{R}^{|V| \times |V|}$ is the diagonal degree matrix such that $\boldsymbol{D}[i, i] = \sum_j \boldsymbol{A}[i, j]$ is the degree of node i and $\boldsymbol{I} \in \mathbb{R}^{|V| \times |V|}$ is the identity matrix.

The graph Laplacian $\boldsymbol{L}$ is an essential operator in spectral graph analysis, whose eigen-decomposition is

$$\boldsymbol{L} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\top, \tag{2.10}$$

where $\boldsymbol{U} = [\boldsymbol{u}_1, ..., \boldsymbol{u}_{|V|}] \in \mathbb{R}^{|V| \times |V|}$ is the matrix of eigenvectors of $\boldsymbol{L}$, and $\boldsymbol{\Lambda} \in \mathbb{R}^{|V| \times |V|}$ is the diagonal matrix of the corresponding eigenvalues $\boldsymbol{\Lambda}[i, i] = \lambda_i$. The *graph Fourier transform* of node-level signal $\boldsymbol{x} \in \mathbb{R}^{|V|}$ is defined as $\boldsymbol{U}^\top \boldsymbol{x}$ and its inverse is $\boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{x})$. Now we are ready to re-write equation 2.9 in terms of the graph Fourier transform:

$$\boldsymbol{x} \star_{\mathcal{G}} \boldsymbol{y} = \boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{x} \odot \boldsymbol{U}^\top \boldsymbol{y}). \tag{2.11}$$

The main design concern when constructing a spectral graph convolution is the choice of the filter $\boldsymbol{U}^\top \boldsymbol{y}$. The spectral filter can be defined as $p_N(\boldsymbol{\Lambda})$, such that the spectral graph convolution becomes

$$\boldsymbol{x} \star_{\mathcal{G}} \boldsymbol{y} = (\boldsymbol{U} p_N(\boldsymbol{\Lambda}) \boldsymbol{U}^\top)\boldsymbol{x} \tag{2.12}$$

$$= (p_N(\boldsymbol{L}))\boldsymbol{x}, \tag{2.13}$$

where $p_N(\boldsymbol{\Lambda}) = \sum_{i=0}^{N-1} \boldsymbol{\theta}_i \boldsymbol{\Lambda}^i$ is a polynomial of eigenvalues of the graph Laplacian of degree $N-1$ and $\boldsymbol{\theta} \in \mathbb{R}^N$ is a vector of polynomial coefficients. The calculation of equation 2.12 is costly, with a time complexity of $\mathcal{O}(|V|^2)$. To reduce the computational overhead, Defferrard et al. [6] use the following approximation:

$$p_N(\boldsymbol{\Lambda}) = \sum_{i=0}^{N-1} \boldsymbol{\theta}_i \boldsymbol{T}_i(\tilde{\boldsymbol{\Lambda}}), \tag{2.14}$$

where $\boldsymbol{T}_i(\tilde{\boldsymbol{\Lambda}}) \in \mathbb{R}^{|V|\times|V|}$ is the recursively defined Chebyshev polynomial for normalized eigenvalues $\tilde{\boldsymbol{\Lambda}} = \frac{2}{\lambda_{\text{max}}}\boldsymbol{\Lambda} - \boldsymbol{I} \in \mathbb{R}^{|V|\times|V|}$ and $\boldsymbol{\theta} \in \mathbb{R}^N$ is a vector of Chebyshev coefficients. Thus the spectral graph convolution is defined as

$$\boldsymbol{x} \star_{\mathcal{G}} \boldsymbol{y} = \left(\boldsymbol{U}\left(\sum_{i=0}^{N-1} \boldsymbol{\theta}_i \boldsymbol{T}_i(\tilde{\boldsymbol{\Lambda}})\right)\boldsymbol{U}^\top\right)\boldsymbol{x} \tag{2.15}$$

$$= \left(\sum_{i=0}^{N-1} \boldsymbol{\theta}_i \boldsymbol{T}_i(\tilde{\boldsymbol{L}})\right)\boldsymbol{x}, \tag{2.16}$$

where $\boldsymbol{T_i}(\tilde{\boldsymbol{L}}) \in \mathbb{R}^{|V|\times|V|}$ is the Chebyshev polynomial for the normalized graph Laplacian $\tilde{\boldsymbol{L}} = \frac{2}{\lambda_{\text{max}}}\boldsymbol{L} - \boldsymbol{I} \in \mathbb{R}^{|V|\times|V|}$. This spectral graph convolution reduces the time complexity to $\mathcal{O}(N|E|)$, which is an improvement in the case of sparse graphs.

The graph convolutional network (GCN) [15] is a first-order spectral graph convolution within the GNN message passing framework. The GCN has the following AGGREGATE and UPDATE functions to calculate the node embedding for node $u \in V$ at iteration $k+1$:

$$\boldsymbol{m}_{\mathcal{N}(u)}^{(k)} = \text{AGGREGATE}\left(\left\{ \boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u) \right\}\right) = \sum_{v \in \mathcal{N}(u)} \frac{1}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}} \boldsymbol{h}_v^{(k)} \tag{2.17}$$

$$\boldsymbol{h}_u^{(k+1)} = \text{UPDATE}\left(\boldsymbol{h}_u^{(k)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}\right) = \sigma\left(\boldsymbol{W}_{\text{self}}^{(k)} \boldsymbol{h}_u^{(k)} + \boldsymbol{W}_{\text{neigh}}^{(k)} \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}\right). \tag{2.18}$$

Note the term $1/\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}$ is a degree normalization constant derived from the normalized Laplacian in definition 2.2.7. Re-written in matrix notation, the GCN message passing function becomes:

$$\boldsymbol{H}^{(k+1)} = \sigma\left(\boldsymbol{H}^{(k)} \boldsymbol{W}_{\text{self}}^{(k)} + \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{H}^{(k)} \boldsymbol{W}_{\text{neigh}}^{(k)}\right), \tag{2.19}$$

where $\boldsymbol{H}^{(k)} \in \mathbb{R}^{|V| \times D^{(k)}}$ is the matrix of all node embeddings at iteration k and $\boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix. The GCN message passing function can be viewed as the non-linear activation of the spectral graph convolution from equation 2.15, in the case where $\lambda_{\text{max}} = 2$ and $N = 2$:

$$\boldsymbol{x} \star_{\mathcal{G}} \boldsymbol{y} = \boldsymbol{\theta}_0 \boldsymbol{x} + \boldsymbol{\theta}_1 (\boldsymbol{L} - \boldsymbol{I}) \boldsymbol{x} \tag{2.20}$$

$$= \boldsymbol{\theta}_0 \boldsymbol{x} + \boldsymbol{\theta}_1 \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{x}. \tag{2.21}$$

The GCN was originally designed for the task of semi-supervised node classification, where a subset of nodes $U \subset V$ is masked and the goal is to learn node embeddings to predict the label of each of the masked nodes with one of $\{ 1, ..., C \}$. To reduce overfitting,

a single parameter variant of the GCN message passing function in equation 2.19 is:

$$\boldsymbol{H}^{(k+1)} = \sigma\left(\left(\boldsymbol{I} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\right)\boldsymbol{H}^{(k)}\boldsymbol{W}^{(k)}\right), \tag{2.22}$$

where $\boldsymbol{W} = \boldsymbol{W}_{\text{self}} = \boldsymbol{W}_{\text{neigh}}$. The final iteration K of message passing yields node embeddings $\boldsymbol{Z} \in \mathbb{R}^{|V| \times F}$. As we will see in section 2.3, the learned node embeddings $\boldsymbol{Z}$ can be used for a multitude of tasks including link prediction, where the task is to predict whether an edge exists between two given nodes.

2.3 Variational Graph Autoencoders

Variational graph autoencoders (VGAEs) are the extension of variational autoencoders (VAEs) to the graph domain. This thesis is built upon VGAEs, which we will see provide convenient node-level embeddings for the task of link prediction. This section begins with a brief overview of generic latent variable models and in particular the VAE as conceived by Kingma and Welling [13]. We proceed by defining the task of link prediction. Finally, we introduce the components of a variational graph autoencoder (VGAE): the graph convolutional network (GCN) encoder, the dot product decoder, and the optimization of the VGAE objective function for end-to-end differentiable learning.

2.3.1 Encoder-decoder Latent Variable Model

The motivation for *latent variable models* is the assumption that there exists both observable variables $\boldsymbol{x} \in \mathbb{R}^D$ and latent (hidden) variables $\boldsymbol{z} \in \mathbb{R}^F$ which are needed to model a data distribution. The goal is to formulate a probabilistic model of the data distribution as

a joint distribution over observable and latent variables:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int_{\boldsymbol{z}} p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} \tag{2.23}$$

$$= \int_{\boldsymbol{z}} p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) d\boldsymbol{z}, \tag{2.24}$$

such that the parameters $\boldsymbol{\theta}$ are learned to maximize $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ for all observable data $\boldsymbol{x}$ in dataset $\mathcal{D}$. Equation 2.23 is called the *marginal likelihood* or the *model evidence*. The distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is referred to as the model *decoder*, which is often modelled with the Gaussian distribution:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \tag{2.25}$$

$$= \mathcal{N}(\boldsymbol{x}; f_{\boldsymbol{\theta}}(\boldsymbol{z}), \sigma^2 \boldsymbol{I}), \tag{2.26}$$

where $f_{\boldsymbol{\theta}}(\boldsymbol{z})$ is a MLP that outputs mean $\boldsymbol{\mu} \in \mathbb{R}^D$. However, the model evidence $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ in equation 2.23 becomes intractable in this case, and thus requires approximate inference techniques to compute. In their construction of the variational autoencoder, Kingma and Welling [13] use *variational inference* to approximate the model *encoder*, also called the model *posterior*:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \tag{2.27}$$

with a variational term $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, which is then used to approximate the otherwise intractable model evidence $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. The model evidence is maximized through the maximization of its

lower bound, referred to as the *evidence lower bound (ELBO)*, which we now derive:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left(\log p_{\boldsymbol{\theta}}(\boldsymbol{x})\right) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right)\right) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} * \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right)\right) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right)\right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right)\right), \qquad (2.28)
\end{aligned}$$

where the ELBO is

$$\begin{aligned}
\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right)\right) \qquad (2.29) \\
&= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - \mathrm{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \,||\, p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) \qquad (2.30) \\
&\leqslant \log p_{\boldsymbol{\theta}}(\boldsymbol{x}), \qquad (2.31)
\end{aligned}$$

and $\mathrm{KL}(p \,||\, q)$ measures the non-negative Kullback-Leibler divergence [17] between the distributions p and q. Here we see maximizing the ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$ over the parameters $\boldsymbol{\theta}, \boldsymbol{\phi}$ maximizes the model evidence $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. Now, let us re-write the ELBO in the form

commonly used to define the VAE objective:

$$\mathcal{L}_{\boldsymbol{\theta},\phi}(\boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right)\right) \tag{2.32}$$

$$= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})) - \log(q_\phi(\boldsymbol{z}|\boldsymbol{x}))\right) \tag{2.33}$$

$$= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log p_{\boldsymbol{\theta}}(\boldsymbol{z}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})\right) \tag{2.34}$$

$$= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log\left(\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right)\right) \tag{2.35}$$

$$= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right) - \mathrm{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \,||\, p_{\boldsymbol{\theta}}(\boldsymbol{z})\right). \tag{2.36}$$

The first term of equation 2.36 is referred to as the *reconstruction* term, which measures how well input $\boldsymbol{x}$ is recovered from its encoded latent $\boldsymbol{z}$. The second term of equation 2.36 regularizes the encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ with a chosen *prior* $p_{\boldsymbol{\theta}}(\boldsymbol{z})$, to prevent the encoder from overfitting the training data. A common choice for the prior is the standard Gaussian $\mathcal{N}(\boldsymbol{z}; \mathbf{0}, \boldsymbol{I})$, however other distributions can be used to regularize the encoder accordingly.

The motivation for encoder-decoder latent variable models is the assumption that there exists a low-dimensional representation of the data we wish to model. The encoder is trained to map high-dimensional input data to a low-dimensional latent space, and the decoder is trained to learn the inverse of this mapping. In short, the goal of an encoder-decoder latent variable model is to learn "meaningful" latent embeddings. The rest of this section explores encoding and decoding graphs at the node level, within the variational graph autoencoder (VGAE).

2.3.2 Link Prediction

Link prediction is a popular task within machine learning and is used in a multitude of applications such as drug reactivity [31] and knowledge base completion [22]. Given a graph $G = (V, E)$ and its adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|V|\times|V|}$ and node features $\boldsymbol{X} \in \mathbb{R}^{|V|\times D}$, the task of *link prediction* is the inference of whether an edge exists between a given pair of nodes. A subset of edges $E_{\text{train}} \subset E$ is used to train the machine learning model, which is then evaluated on the test set $E_{\text{test}} = (E - E_{\text{train}}) \cup E'$, where $(E - E_{\text{train}})$ is the set of edges existing in the original graph and E' is the set of non-existent edges.

Link prediction is considered a semi-supervised task because the model is given access to a subset of edges during training, rather than to the full graph as in the supervised setting. As previously mentioned in our discussion of simplicial complexes in section 2, the set of edges E represents a binary relation. Thus, the task of link prediction can be viewed as a specific case of *relation prediction*, the general task of inferring relations of a given order.

2.3.3 Encoder

Let us consider graph $G = (V, E)$, with its structure represented by adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|V|\times|V|}$ and its node features $\boldsymbol{X} \in \mathbb{R}^{|V|\times D}$. In a VGAE, the *encoder* $q_{\phi} : \left(\mathbb{R}^{|V|\times D}\right) \times \left(\mathbb{R}^{|V|\times|V|}\right) \to \text{Dist}\left(\mathbb{R}^{|V|\times F}\right)$ maps the node features $\boldsymbol{X} \in \mathbb{R}^{|V|\times D}$ to a distribution over node latents $\boldsymbol{Z} \in \mathbb{R}^{|V|\times F}$, where $\boldsymbol{X}[u] \in \mathbb{R}^{D}$ and $\boldsymbol{Z}[u] \in \mathbb{R}^{F}$ are the node features and the node

latent for node u, respectively. The encoder is modelled with the Gaussian distribution:

$$q_{\phi}(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{A}) = \prod_{i=1}^{|V|} q_{\phi,i}\left(\boldsymbol{Z}[i] \mid \boldsymbol{X}, \boldsymbol{A}\right) \tag{2.37}$$

$$= \prod_{i=1}^{|V|} \mathcal{N}\left(\boldsymbol{Z}[i]; \boldsymbol{\mu}[i], \text{diag}(\boldsymbol{\sigma}^2[i])\right), \tag{2.38}$$

where the assumption of conditional independence is used in equation 2.37. How does one learn the parameters $\boldsymbol{\phi}$ such that the encoder distribution $q_{\phi}(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{A})$ is maximized for given node features $\boldsymbol{X}$ and adjacency matrix $\boldsymbol{A}$? Kipf et al. [14] select a 2-layer graph convolutional network (GCN) as seen in equation 2.22:

$$\boldsymbol{\mu} = \text{GCN}_{\boldsymbol{\mu}}(\boldsymbol{X}, \boldsymbol{A}) = \tilde{\boldsymbol{A}}\text{ReLU}(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}_{\boldsymbol{\mu}_0})\boldsymbol{W}_{\boldsymbol{\mu}_1}, \tag{2.39}$$

$$\log(\boldsymbol{\sigma}) = \text{GCN}_{\boldsymbol{\sigma}}(\boldsymbol{X}, \boldsymbol{A}) = \tilde{\boldsymbol{A}}\text{ReLU}(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}_{\boldsymbol{\sigma}_0})\boldsymbol{W}_{\boldsymbol{\sigma}_1}, \tag{2.40}$$

where $\boldsymbol{\mu} \in \mathbb{R}^{|V| \times F}$, $\boldsymbol{\sigma} \in \mathbb{R}^{|V| \times F}$, $\tilde{\boldsymbol{A}} = \boldsymbol{I} + \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix and $\boldsymbol{D} \in \mathbb{R}^{|V| \times |V|}$ is the diagonal degree matrix as seen in definition 2.2.7. Here, the GCN can be replaced with any graph neural network (GNN) which utilizes both the input graph structure and node features. Finally, once the encoder is trained, node latents $\boldsymbol{Z}$ are sampled from the learned encoder distribution $q_{\phi}(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{A})$. In order to avoid overfitting, the encoder is regularized with the prior $p(\boldsymbol{Z}) = \mathcal{N}(\boldsymbol{Z}; \boldsymbol{0}, \boldsymbol{I})$.

There exists a non-probabilistic version of the VGAE, simply referred to as the *graph autoencoder* (GAE). In this setting, the encoder is a deterministic function learned by a

graph neural network:

$$\boldsymbol{Z} = \text{GCN}(\boldsymbol{X}, \boldsymbol{A}) = \tilde{\boldsymbol{A}}\text{ReLU}(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}_0)\boldsymbol{W}_1. \tag{2.41}$$

In practice, the GAE is more susceptible to overfitting than the VGAE. As detailed in chapter 3, this thesis builds upon both the GAE and VGAE.

2.3.4 Decoder

Given a set of node-level latents $\boldsymbol{Z} \in \mathbb{R}^{|V| \times F}$ generated by the encoder, the VGAE decoder $p_{\boldsymbol{\theta}} : \mathbb{R}^{|V| \times F} \rightarrow \text{Dist}(\mathbb{R}^{|V| \times |V|})$ maps the node-level latents to a distribution over adjacency matrices $\boldsymbol{A} \in \mathbb{R}^{|V| \times |V|}$. The goal of the decoder is to reconstruct the structure of the original input graph, using the encoded node latents. The decoder takes the form:

$$p_{\boldsymbol{\theta}}(\boldsymbol{A} \mid \boldsymbol{Z}) = \prod_{i=1}^{|V|} \prod_{j=1}^{|V|} p_{\boldsymbol{\theta}}(\boldsymbol{A}[i,j] = 1 \mid \boldsymbol{Z}[i], \boldsymbol{Z}[j]) \tag{2.42}$$

$$= \prod_{i=1}^{|V|} \prod_{j=1}^{|V|} = l(s(\boldsymbol{Z}[i], \boldsymbol{Z}[j])), \tag{2.43}$$

where $s : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}$ is a scoring function for the pair of node latents and $l : \mathbb{R} \rightarrow [0, 1]$ is a nonlinear activation. Note in equation 2.42 the assumption of independence between edges allows for the factorization over Bernoulli distributions. In their VGAE formulation, Kipf et al. [14] select a simple dot-product scoring function and sigmoid nonlinear activation to model the Bernoulli distribution for edge (i, j):

$$p_{\boldsymbol{\theta}}(\boldsymbol{A}[i,j] = 1 \mid \boldsymbol{Z}[i], \boldsymbol{Z}[j]) = \sigma(\boldsymbol{Z}[i]^\top \boldsymbol{Z}[v]). \tag{2.44}$$

This scoring function does not have learned parameters, although more expressive parameterized decoders that are valid probability distributions can be employed.

For the non-probabilistic GAE, Kipf et al. [14] use the same scoring function and nonlinear activation as in equation 2.44, except the output does not model a Bernoulli distribution and instead corresponds to a soft adjacency matrix $\hat{\boldsymbol{A}} \in \mathbb{R}^{|V|\times|V|}$:

$$\hat{\boldsymbol{A}}[i,j] = \sigma(\boldsymbol{Z}[i]^\top \boldsymbol{Z}[j]), \tag{2.45}$$

which we will see in section 2.3.5 can be trained using a binary cross-entropy loss.

2.3.5 Learning

Learning, the process of iteratively updating parameters to maximize a given objective function, is where the benefit of node-level latents becomes apparent. Based on equation 2.36, the objective function for the VGAE is the ELBO for graphs:

$$\mathcal{L}_{\phi} = \mathbb{E}_{q_{\phi}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}\left(\log p(\boldsymbol{A} \mid \boldsymbol{Z})\right) - \mathrm{KL}\left(q_{\phi}(\boldsymbol{Z} \mid \boldsymbol{X},\boldsymbol{A}) \,||\, p(\boldsymbol{Z})\right). \tag{2.46}$$

Note that distribution p is not parameterized, although the VAE allows for this. The adjacency matrix $\boldsymbol{A}$ the decoder reconstructs from the given node latents $\boldsymbol{Z}$ is binary-valued and thus the reconstruction term $\mathbb{E}_{q_{\phi}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}\left(\log p(\boldsymbol{A} \mid \boldsymbol{Z})\right)$ is calculated using a binary-cross entropy loss of the following form:

$$\mathcal{L}_{\text{recon}} = -\frac{1}{|V|^2}\sum_{i=1}^{|V|}\sum_{j=1}^{|V|} \boldsymbol{A}[i,j]\log l(s(\boldsymbol{Z}[i],\boldsymbol{Z}[j])) + (1-\boldsymbol{A}[i,j])\log\left(1 - l(s(\boldsymbol{Z}[i],\boldsymbol{Z}[j]))\right). \tag{2.47}$$

Note the use of node-level latents circumvents the problem of graph isomorphism: input node u corresponds to its latent representation and reconstruction, thus enabling a straightforward calculation of the binary cross-entropy loss. This is a significant challenge for autoencoders which encode latent representations at the graph level, such as GraphVAE [27], where no feasible exact matching of the input and output nodes exists.

Now let us consider how to optimize the variational parameters $\boldsymbol{\phi}$. A problem occurs when trying to compute the gradient of the ELBO with respect to $\boldsymbol{\phi}$:

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\phi}} = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})} \left(\log p(\boldsymbol{X}, \boldsymbol{A}, \boldsymbol{Z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{A}) \right) \tag{2.48}$$

$$\neq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})} \left(\nabla_{\boldsymbol{\phi}} \left(\log p(\boldsymbol{X}, \boldsymbol{A}, \boldsymbol{Z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{A}) \right) \right). \tag{2.49}$$

To overcome this problem, a technique known as the *reparameterization trick* [13] [24] is used to isolate the stochasticity when sampling from the encoder $q_{\boldsymbol{\phi}}(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{A})$:

$$\boldsymbol{Z} = \boldsymbol{\epsilon} \odot \boldsymbol{\sigma} + \boldsymbol{\mu}, \tag{2.50}$$

$$\text{where } \boldsymbol{\epsilon} \sim f(\boldsymbol{\epsilon}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$$

and $\boldsymbol{\phi} = \{ \boldsymbol{\mu}, \boldsymbol{\sigma} \}$ are the parameters for which we want to calculate the gradient of the ELBO. The gradient of the ELBO can now be written as:

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\phi}} = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})} \left(\log p(\boldsymbol{X}, \boldsymbol{A}, \boldsymbol{Z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{A}) \right) \tag{2.51}$$

$$= \mathbb{E}_{f(\boldsymbol{\epsilon})} \left(\nabla_{\boldsymbol{\phi}} \left(\log p(\boldsymbol{X}, \boldsymbol{A}, \boldsymbol{Z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{A}) \right) \right), \tag{2.52}$$

which can be estimated using Monte Carlo approximation.

For the GAE, the objective function consists solely of the reconstruction term as there is no need to regularize the deterministic encoder. The reconstruction term is computed with the binary cross-entropy loss seen in equation 2.47 and the reparametrization trick is irrelevant in this non-stochastic setting.

2.4 Topological Data Analysis

Topological data analysis is a field of study largely concerned with the shape of data; in particular the presence of holes. The simplicial complex is a useful mathematical construct to generalize networks to higher order relations, in a geometric space. We explore two methods of construction of simplicial complexes: the Čech complex and the Vietoris-Rips complex. Each of these constructions maps a set of points embedded in a metric space to an abstract simplicial complex. Let us begin by defining a metric space.

Definition 2.4.1 (Metric space (X, d))**.** *A* metric space *is a set X equipped with a function $d : X \times X \to \mathbb{R}^{\geqslant 0}$ satisfying:*

1. *For all $x \in X$, $d(x, x) = 0$,* *(reflexivity)*
2. *For all $x, y \in X$, $d(x, y) = d(y, x)$,* *(symmetry)*
3. *For all $x, y, z \in X$, $d(x, y) \leqslant d(x, z) + d(z, y)$,* *(triangle inequality)*
4. *For all $x, y \in X, d(x, y) = 0$ if and only if $x = y$.*

We call d a metric *and the pair (X, d) a* metric space.

2.4.1 Čech Complex

The Čech complex determines connectivity based on circles centered at each embedded point in the metric space, thus connecting simplicial complexes with metric spaces [10].

Definition 2.4.2 (Čech complex)**.** *Let us consider a set of* $V = \{ v_0, v_1, ..., v_N \}$ *vertices embedded in the metric space* $(\mathbb{R}^n, d)$ *and radius* $r > 0$*. The* Čech complex *is the construction of the abstract simplicial complex* K *equal to the union of every* k*-simplex such that:*

$$\bigcap_{i=0}^{k} B(v_i, r) \neq \varnothing,$$

where $B(v_i, r)$ *is a ball of radius* r *centered at vertex* v_i *and* k *ranges over the values* $\{ 0, 1, \ldots, N \}$.

Note that the edges (1-simplices) of a triangle might be present in Čech complex K but the 2-simplex triangle is not necessarily an element of K; the 2-simplex triangle also requires the triple intersection of balls centered at the corresponding nodes be non-empty. Figure 2.3 visualizes the construction of a Čech complex from a set of embedded vertices. The hierarchical construction of Čech complexes across dimensions ensures it satisfies the properties of an abstract simplicial complex in definition 2.1.7.

2.4.2 Vietoris-Rips Complex

The Vietoris-Rips complex is an alternate and more computationally efficient construction than the Čech complex [9].

Definition 2.4.3 (Vietoris-Rips complex)**.** *Consider a set of* $V = \{ v_0, v_1, \ldots, v_N \}$ *vertices embedded in the metric space* $(\mathbb{R}^n, d)$ *and radius* $r > 0$*. The* Vietoris-Rips complex *is the construction of the abstract simplicial complex* K *equal to the union of every* k*-simplex* σ *such that:*

$$d(v_i, v_j) \leqslant r \textit{ for all } v_i, v_j \in \sigma.$$

The metric d is often chosen to be the Euclidean metric, which has the following familiar definition.

Definition 2.4.4 (Euclidean metric d)**.** *Let* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$*. Then the* Euclidean metric *is*

$$d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\| = \sqrt{\sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{y}_i)^2}.$$

Note that the Vietoris-Rips complex is, like the Čech complex, also a valid abstract simplicial complex. Unlike the Čech complex, if the edges (1-simplices) of a triangle are present in the Vietoris-Rips complex K, the 2-simplex triangle is necessarily an element of K. Illustrated in Figure 2.3, if each pair of 3 vertices is within r distance, then by definition the three edges and triangle connecting these vertices must be present. This is an important consideration when selecting a construction for relation prediction, which is discussed in section 3.4.

Both of these types of simplicial complexes are often used for *persistent homology* [5], the study of the appearance and disappearance of topological holes in data as parameters are varied. However, this thesis is concerned with the quality of information captured by node embeddings, measured by how well a reconstructed simplicial complex matches the

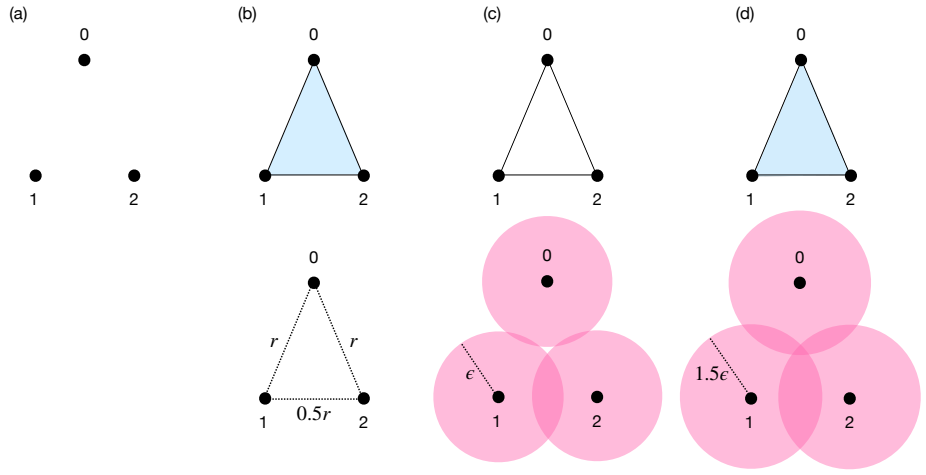


Figure 2.3: (a) is a set of vertices embedded in $\mathbb{R}^n$ and its corresponding simplicial complex constructions: (b) Vietoris-Rips complex constructed from distance r and the distances between each pair of embedded vertices, (c) Čech complex constructed from balls of radius ϵ, (d) Čech complex constructed from balls of radius 1.5ϵ. Čech complex (c) has an empty intersection of the balls centered at the embedded vertices and is thus a 1-dimensional simplicial complex Čech complex, whereas (d) has a non-empty intersection and is thus a 2-dimensional simplicial complex.

input simplicial complex.

3

Variational Autoencoders for Simplicial Complexes

This chapter weaves the extensive work done on simplicial complexes with variational autoencoders to produce a novel variational autoencoder suitable for relation prediction, the generalization of link prediction to higher order relations. In this thesis we focus on the extension of the variational graph autoencoder to the variational autoencoder for 2-dimensional simplicial complexes, however, our formulation can be extended to higher dimensional simplicial complexes. This chapter begins with our representation of abstract simplicial complexes, which combines computationally efficient simplex trees [4] with a tensor representation amenable to machine learning frameworks. We then present the encoders, comprised of a message passing neural network for simplicial complexes. Next we present the decoders, a multi-linear dot product and a differentiable variant of the Čech complex. Finally, we directly define the optimization objective for the variational simplicial autoencoder, without relying on additional regularization terms.

The novelty of this thesis lies in the adaptation of the variational graph autoencoder to the variational autoencoder for simplicial complexes. Our goal is to encode meaningful node-level embeddings, which in turn are used to infer relations between entities. This thesis limits its focus to latent variable models, although other models designed for higher order link prediction on hypergraphs [29] [18] are comparable. Previous work has been done on autoencoders for simplicial complexes, however these works differ both in approach and application: simplicial autoencoders [8] constrain the latent embeddings to a simplicial complex via a regularizer and this model is designed for image-domain tasks, rather than link prediction; topological autoencoders [21] add a topological-preserving regularizer to the autoencoder objective; autoencoding topology in generative models [16] addresses manifold learning by viewing the decoder as an atlas, a set of maps, to be learned from the latent embeddings. We now present our variational autoencoder for simplicial complexes.

3.1 Simplicial Complex Representation

Abstract simplicial complexes are combinatorial objects on which operations can quickly become infeasible. The design goal of our simplicial complex representation is two-fold:

1. Computationally efficient representation for simplicial complex operations.
2. Representation compatible with a widely used graph representation learning framework.

The first design concern is addressed with simplex trees in section 3.1.1. The second concern factors into the design of a tensor representation for simplicial complexes in section 3.1.2.

3.1.1 Simplex Trees

Simplex trees are a data structure designed by Boissonnat et al. [4] to efficiently store abstract simplicial complexes of any finite dimension. The crucial property of simplex trees is the storing of one node per simplex, which enables the efficient storage of incident relations between simplices.

Definition 3.1.1 (Simplex tree of K)**.** *Let K be an abstract simplicial complex with vertices $V = \{ v_0, v_1, \ldots, v_N \}$, where each k-simplex of K is a sequence $\sigma = \{ v_{l_0}, v_{l_1}, \ldots, v_{l_k} \}$, $v_{l_i} \in V$, $l_i \in \{ 0, \ldots, N \}$, and $l_0 < l_1 < \cdots < l_k$. The* simplex tree of K *is a tree $\mathcal{T}$ satisfying the following properties:*

1. *The nodes of the simplex tree are in bijection with the constituent simplices of K. The root of the simplex tree is associated with the empty simplex $\sigma = \varnothing$.*
2. *Each node of the simplex tree, except the root, stores the label of the last vertex in the k-simplex sequence σ to which the node is associated, where last$(\sigma) = l_k$.*
3. *The vertices whose labels are encountered along a path from the root of the simplex tree to a node n, associated with a simplex σ, are the vertices of σ. The labels are sorted by increasing order along such a path, and each vertex label appears exactly once.*

Note the ordering of vertices in a sequence is necessary to attain a bijection between simplex tree nodes and simplices. We illustrate with an example in Figure 3.1.

Let us consider the runtime of some important operations on simplex trees. Let D_m be the maximal number of operations needed to perform a search, insertion, or removal of a

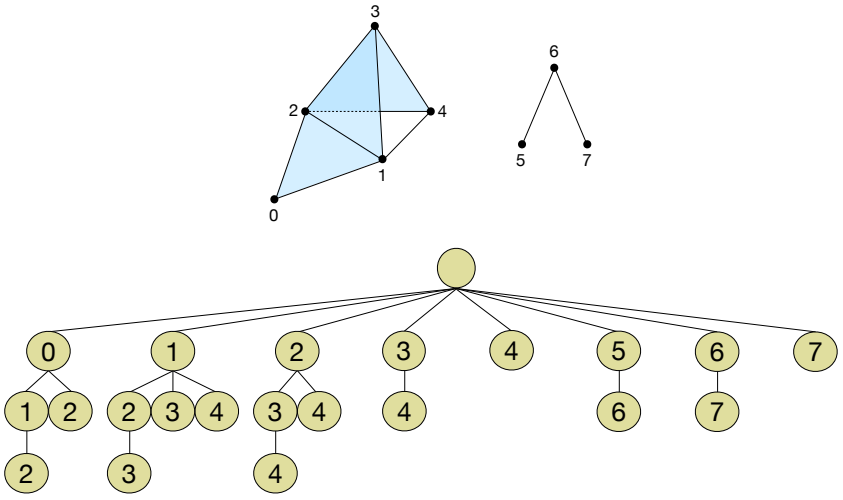


Figure 3.1: Above is a 2-dimensional abstract simplicial complex and below is its simplex tree representation. The nodes at depth d in the simplex tree correspond to $(d-1)$-dimensional simplices, except the root. This simplicial complex has 8 0-simplices (vertices), 10 1-simplices (edges), and 3 2-simplices (triangles).

node in simplex tree $\mathcal{T}$. The term D_m depends on two factors: the maximal degree of a node, distinct from the root, in simplex tree $\mathcal{T}$, and the type of tree data structure used to represent $\mathcal{T}$. Assume we wish to find a k-simplex σ in the simplex tree $\mathcal{T}$. This operation takes time $\mathcal{O}((k+1)D_m)$ as a path of size $k+1$ must be found from the root to a node n, associated with the vertices of σ. This operation can be extended to insert a k-simplex σ and all of its lower dimensional faces in the simplex tree $\mathcal{T}$. The k-simplex σ has 2^{k+1} subfaces and thus recursive insertion requires time $\mathcal{O}(2^{k+1}D_m)$.

The removal of a simplex requires the removal of its cofaces to ensure the simplex tree still represents a valid simplicial complex. Assume we wish to remove the simplex $\sigma = \left\{ v_{l_0}, v_{l_1}, \ldots, v_{l_j} \right\}$ from simplex tree $\mathcal{T}$ representing simplicial complex K. Cofaces of simplex σ take the form $\left\{ \star v_{l_0} \star v_{l_1} \star \cdots \star v_{l_j} \star \right\}$, where $\star$ represents an arbitrary subsequence. First, all simplices of the form $\left\{ \star v_{l_0} \star v_{l_1} \star \cdots \star v_{l_j} \right\}$ are searched: if a node N_{l_j}

is found at depth at least $j+1$, then the tree $\mathcal{T}$ is traversed upward to the root, yielding cofaces ending in label l_j. This procedure takes $\mathcal{O}(k)$ time, where k is the dimension of simplicial complex K. The remaining cofaces of simplex σ are represented by the nodes in the subtree rooted at N_{l_j}. Thus the complexity of finding all the cofaces of simplex σ is $\mathcal{O}(k\mathcal{T}_{l_j}^{>j})$, where $\mathcal{T}_{l_j}^{>j}$ is the number of nodes with label l_j at depth at least $j+1$ in simplex tree $\mathcal{T}$.

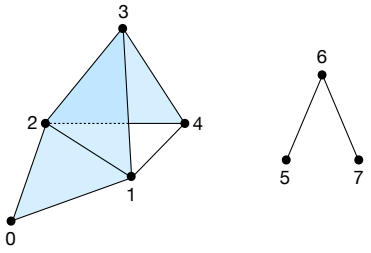
The Gudhi (Geometric Understanding in Higher Dimensions) library [20] provides a Python interface on top of an efficient C++ backend for state-of-the-art computational topology algorithms. This thesis employs Gudhi simplex trees as an efficient data structure to create and update valid simplicial complexes.

3.1.2 Tensor Representation

The simplex tree is an efficient data structure for simplicial complex operations, however, a tensor representation is required for representation learning compatible with the widely used software library PyTorch Geometric [7]. An extension of the deep learning framework PyTorch [23], PyTorch Geometric is designed expressly for efficient representation learning on irregular graphs. In particular, PyTorch Geometric provides a general interface for graph neural networks and message passing. Within PyTorch Geometric, graph $G = (V, E)$ is represented using the coordinate list (COO) as seen in definition 2.2.2. We extend this representation to abstract simplicial complexes.

Definition 3.1.2 (Tensor representation of abstract simplicial complex K)**.** *The* tensor representation *of k-dimensional abstract simplicial complex $K = K_0 \cup \cdots \cup K_k$, is the enumeration of each set K_i for $i \in \{0, \ldots, k\}$.*

(a)



(b)

0	0	1	1	1	2	2	3	5	6	1	2	2	3	4	3	4	4	6	7
1	2	2	3	4	3	4	4	6	7	0	0	1	1	1	2	2	3	5	6

(c)

0	1	2	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4	
1	2	3	2	0	2	0	1	3	1	3	1	2	4	2	4	2	3	
2	3	4	1	2	0	1	0	2	3	1	2	1	3	4	2	3	2	

Figure 3.2: (a) is an abstract simplicial complex $K = K_0 \cup K_1 \cup K_2$ with $|K_0| = 8$, $|K_1| = 10$, and $|K_2| = 3$. The tensor representation of abstract simplicial complex K is composed of: (b) the coordinate list $\boldsymbol{E} \in \mathbb{R}^{2\times 2|K_1|}$, and (c) the coordinate list $\boldsymbol{T} \in \mathbb{R}^{3\times 6|K_2|}$.

Figure 3.2 is an illustration of the tensor representation of abstract simplicial complex K. Note that K_0, K_1, K_2 are the sets of vertices, edges, and triangles, respectively. In practice, our abstract simplicial complex representation is a PyTorch Geometric data class `AbstractSimplicialComplex` of our tensor representation in definition 3.1.2, with an additional simplex tree property. This property is used to perform simplicial complex operations, whereas the tensor representation is used for representation learning. Within the class `AbstractSimplicialComplex`, we provide methods to convert to and from simplex tree and tensor representations. Equipped with an abstract simplicial complex representation, we now present the representation learning task relation prediction.

3.2 Relation Prediction

In this thesis we define relation prediction as the generalization of link prediction to higher order relations. Of particular use in domains where we wish to model relations between three or more entities, applications of relation prediction include co-authorship citation networks, online thread participation, and drug networks [3].

In the link prediction setting discussed in section 2.3.2, an adjacency matrix represents the graph's structure and a node feature matrix captures node-level features. However, in the relation prediction setting, there are higher order features to be considered. Let $K = K_0 \cup K_1 \cup ... \cup K_k$ be a k-dimensional abstract simplicial complex with node features $\boldsymbol{X}_0 \in \mathbb{R}^{|K_0| \times D_0}$, edge features $\boldsymbol{X}_1 \in \mathbb{R}^{|K_1| \times D_1}$, triangle features $\boldsymbol{X}_2 \in \mathbb{R}^{|K_2| \times D_2}$, and higher order features up to $\boldsymbol{X}_k \in \mathbb{R}^{|K_k| \times D_k}$, where D_0, D_1, D_2, and D_k are the number of features for each node, edge, triangle, and k-simplex respectively. The task of *relation prediction* is the inference of whether a j-simplex exists, given a set of $(j-1)$-simplices and their corresponding features. This thesis explores the task of triangle prediction. For this task, a subset of triangles $(K_2)_{\text{train}} \subset K_2$ is used to train the machine learning model, which is then evaluated on the test set $(K_2)_{\text{test}} = (K_2 - (K_2)_{\text{train}}) \cup K_2'$, where $(K_2 - (K_2)_{\text{train}})$ is the set of existing triangles in simplicial complex K and K_2' is the set of non-existent triangles.

3.3 Encoder

The aim of the encoder is to compress hierarchical relations and their corresponding features into node-level latent embeddings. These node-level embeddings circumvent the problem of simplicial complex isomorphism when calculating the reconstruction loss, as will

be detailed in section 3.5. We extend the message passing framework to abstract simplicial complexes; the higher order counterpart to graph neural networks.

3.3.1 Message Passing for Simplicial Complexes

Message passing for 2-dimensional abstract simplicial complex $K = K_0 \cup K_1 \cup K_2$ has three steps:

1. Update triangle embeddings of the form $\boldsymbol{h}_{\{\,u,v,w\,\}}$, where $\{\,u\,\}, \{\,v\,\}, \{\,w\,\} \in K_0$, $\{\,u,v\,\}, \{\,v,w\,\}, \{\,w,u\,\} \in K_1$, and $\{\,u,v,w\,\} \in K_2$.
2. Update edge embeddings of the form $\boldsymbol{h}_{\{\,u,v\,\}}$, where $\{\,u\,\}, \{\,v\,\} \in K_0$ and $\{\,u,v\,\} \in K_1$.
3. Update node embeddings of the form $\boldsymbol{h}_{\{\,u\,\}}$ where $\{\,u\,\} \in K_0$.

Message passing for triangle $\{\,u,v,w\,\} \in K_2$ at iteration $k+1$ is as follows:

$$\boldsymbol{h}^{(k+1)}_{\{\,u,v,w\,\}} = \text{UPDATE}_{\boldsymbol{T}}\left(\boldsymbol{h}^{(k)}_{\{\,u,v,w\,\}}, \boldsymbol{h}^{(k)}_{\{\,u,v\,\}}, \boldsymbol{h}^{(k)}_{\{\,v,w\,\}}, \boldsymbol{h}^{(k)}_{\{\,w,u\,\}}\right) \tag{3.1}$$

$$= \sigma\left(\boldsymbol{W}^{(k)}_{T1}\boldsymbol{h}^{(k)}_{\{\,u,v,w\,\}} + \boldsymbol{W}^{(k)}_{T2}\left(\boldsymbol{h}^{(k)}_{\{\,u,v\,\}} + \boldsymbol{h}^{(k)}_{\{\,v,w\,\}} + \boldsymbol{h}^{(k)}_{\{\,w,u\,\}}\right)\right), \tag{3.2}$$

where $\boldsymbol{h}^{(k)}_{\{\,u,v,w\,\}} \in \mathbb{R}^{D_2{}^{(k)}}$, $\boldsymbol{h}^{(k)}_{\{\,u,v\,\}}, \boldsymbol{h}^{(k)}_{\{\,v,w\,\}}, \boldsymbol{h}^{(k)}_{\{\,w,u\,\}} \in \mathbb{R}^{D_1{}^{(k)}}$, $\boldsymbol{W}_{T1}$ and $\boldsymbol{W}_{T2}$ are each a learnable matrix of parameters, and σ is a non-linear activation such as ReLU. After the triangle embeddings are updated, message passing is performed for each edge $\{\,u,v\,\} \in K_1$. First, let us build upon the notion of a neighbourhood, introduced in definition 2.2.4.

Definition 3.3.1 (j-Neighbourhood of $(j-1)$-simplex s, $\mathcal{N}_j(s)$)**.** *Given the abstract simplicial complex $K = K_0 \cup K_1 \cup ... \cup K_k$, the* j-neighbourhood *of $(j-1)$-simplex $s \in K$ is the set of j-cofaces of s.*

Concretely, the 2-neighbourhood of edge $\{ u, v \}$ is:

$$\mathcal{N}_2(\{ u, v \}) = \{ \{ u, v, w \} \mid \{ u, v \} \in K_1, \{ u, v, w \} \in K_2 \}, \tag{3.3}$$

whereas the 1-neighbourhood of node $\{ u \}$ is:

$$\mathcal{N}_1(\{ u \}) = \{ \{ u, v \} \mid \{ u \} \in K_0, \{ u, v \} \in K_1 \}. \tag{3.4}$$

We are ready to define message passing for each edge $\{ u, v \} \in K_1$:

$$\boldsymbol{m}^{(k+1)}_{\mathcal{N}_2(\{ u,v \})} = \text{AGGREGATE}_{\mathcal{N}_2} \left(\left\{ \boldsymbol{h}^{(k+1)}_{\{ u,v,w \}} \;\middle|\; \{ u, v, w \} \in \mathcal{N}_2(\{ u, v \}) \right\} \right) \tag{3.5}$$

$$= \sum_{\{ u,v,w \} \in \mathcal{N}_2(\{ u,v \})} \boldsymbol{h}^{(k+1)}_{\{ u,v,w \}}, \tag{3.6}$$

$$\boldsymbol{h}^{(k+1)}_{\{ u,v \}} = \text{UPDATE}_E \left(\boldsymbol{h}^{(k)}_{\{ u,v \}}, \boldsymbol{m}^{(k+1)}_{\mathcal{N}_2(\{ u,v \})} \right) \tag{3.7}$$

$$= \sigma \left(\boldsymbol{W}^{(k)}_{E1} \boldsymbol{h}^{(k)}_{\{ u,v \}} + \boldsymbol{W}^{(k)}_{E2} \boldsymbol{m}^{(k+1)}_{\mathcal{N}_2(\{ u,v \})} \right). \tag{3.8}$$

Finally, message passing is performed on each node $\{ u \} \in K_0$:

$$\boldsymbol{m}^{(k+1)}_{\mathcal{N}_1(\{ u \})} = \text{AGGREGATE}_{\mathcal{N}_1} \left(\left\{ \boldsymbol{h}^{(k+1)}_{\{ u,v \}} \;\middle|\; \{ u, v \} \in \mathcal{N}_1(\{ u \}) \right\} \right) \tag{3.9}$$

$$= \sum_{\{ u,v \} \in \mathcal{N}_1(\{ u \})} \boldsymbol{h}^{(k+1)}_{\{ u,v \}}, \tag{3.10}$$

$$\boldsymbol{h}^{(k+1)}_{\{ u \}} = \text{UPDATE}_N \left(\boldsymbol{h}^{(k)}_{\{ u \}}, \boldsymbol{m}^{(k+1)}_{\mathcal{N}_1(\{ u \})} \right) \tag{3.11}$$

$$= \sigma \left(\boldsymbol{W}^{(k)}_{N1} \boldsymbol{h}^{(k)}_{\{ u \}} + \boldsymbol{W}^{(k)}_{N2} \boldsymbol{m}^{(k+1)}_{\mathcal{N}_1(\{ u \})} \right), \tag{3.12}$$

where $\boldsymbol{h}^{(k)}_{\{ u \}} \in \mathbb{R}^{D_0{}^{(k)}}$. Here, each d-simplex embedding is initialized to $\boldsymbol{h}^{(0)}_s = \boldsymbol{X}_d[s] \in$

$\mathbb{R}^{D_d}$.

Message passing for abstract simplicial complexes effectively incorporates triangle, edge, and node features to generate hierarchical embeddings for the various components of the abstract simplicial complex. Note the aggregate and update functions for each dimension may be arbitrary differentiable set functions. In our formulation, we have chosen the same functions as those seen in equations 2.5a and 2.5b.

3.3.2 Probabilistic Autoencoder

Let us consider abstract simplicial complex $K = K_0 \cup K_1 \cup K_2$, with its structure represented by tensors $\boldsymbol{E} \in \mathbb{R}^{2\times|K_1|}, \boldsymbol{T} \in \mathbb{R}^{3\times 6|K_2|}$ and its features $\boldsymbol{X}_0 \in \mathbb{R}^{|K_0|\times D_0}$, $\boldsymbol{X}_1 \in \mathbb{R}^{|K_1|\times D_1}$, $\boldsymbol{X}_2 \in \mathbb{R}^{|K_2|\times D_2}$. In a simplicial variational autoencoder (SVAE), the *encoder* $q_{\boldsymbol{\phi}} : \left(\mathbb{R}^{2\times|K_1|}\right) \times \left(\mathbb{R}^{3\times 6|K_2|}\right) \times \left(\mathbb{R}^{|K_0|\times D_0}\right) \times \left(\mathbb{R}^{|K_1|\times D_1}\right) \times \left(\mathbb{R}^{|K_2|\times D_2}\right) \to \text{Dist}\left(\mathbb{R}^{|K_0|\times F}\right)$ maps the node features to a distribution over node latents $\boldsymbol{Z} \in \mathbb{R}^{|K_0|\times F}$, where $\boldsymbol{X}_k[u] \in \mathbb{R}^{D_k}$ and $\boldsymbol{Z}[u] \in \mathbb{R}^F$ are the node features and the node latent for node u, respectively. The encoder is modelled with the Gaussian distribution:

$$q_{\boldsymbol{\phi}}\left(\boldsymbol{Z}|\boldsymbol{E},\boldsymbol{T},\boldsymbol{X}_0,\boldsymbol{X}_1,\boldsymbol{X}_2\right) = \prod_{i=1}^{|K_0|} q_{\boldsymbol{\phi},i}\left(\boldsymbol{Z}[i] \mid \boldsymbol{E},\boldsymbol{T},\boldsymbol{X}_0,\boldsymbol{X}_1,\boldsymbol{X}_2\right) \tag{3.13}$$

$$= \prod_{i=1}^{|K_0|} \mathcal{N}\left(\boldsymbol{Z}[i]; \boldsymbol{\mu}[i], \text{diag}(\boldsymbol{\sigma}^2[i])\right). \tag{3.14}$$

How does one learn the parameters $\boldsymbol{\phi}$ such that the encoder distribution $q_{\boldsymbol{\phi}}\left(\boldsymbol{Z}|\boldsymbol{E},\boldsymbol{T},\boldsymbol{X}_0,\boldsymbol{X}_1,\boldsymbol{X}_2\right)$ is maximized for given simplex features and simplicial complex structure? We choose a 2-

layer simplicial neural network (message passing) architecture:

$$\boldsymbol{\mu}[i] = \text{SNN}_{\boldsymbol{\mu}}\left(\boldsymbol{E}, \boldsymbol{T}, \boldsymbol{X}_0, \boldsymbol{X}_1, \boldsymbol{X}_2\right) \tag{3.15}$$

$$= \text{ReLU}\left(\boldsymbol{W}^{(1)}_{\boldsymbol{\mu}_{N1}} \boldsymbol{h}^{(1)}_{\{i\}} + \boldsymbol{W}^{(1)}_{\boldsymbol{\mu}_{N2}} \boldsymbol{m}^{(2)}_{\boldsymbol{\mu}_{\mathcal{N}(\{i\})}}\right), \tag{3.16}$$

$$\log(\boldsymbol{\sigma}[i]) = \text{SNN}_{\boldsymbol{\sigma}}\left(\boldsymbol{E}, \boldsymbol{T}, \boldsymbol{X}_0, \boldsymbol{X}_1, \boldsymbol{X}_2\right) \tag{3.17}$$

$$= \text{ReLU}\left(\boldsymbol{W}^{(1)}_{\boldsymbol{\sigma}_{N1}} \boldsymbol{h}^{(1)}_{\{i\}} + \boldsymbol{W}^{(1)}_{\boldsymbol{\sigma}_{N2}} \boldsymbol{m}^{(2)}_{\boldsymbol{\sigma}_{\mathcal{N}(\{i\})}}\right). \tag{3.18}$$

The intent of this architecture is to avoid over-smoothing, where node-level features are homogeneous due to too many iterations of message passing.

3.3.3 Non-probabilistic Autoencoder

We present a non-probabilistic version of the SVAE in section 3.3.2, simply referred to as the *simplicial autoencoder* (SAE). In this setting, the encoder is a deterministic function over the set of nodes $i \in |K_0|$ learned by a simplicial neural network:

$$\boldsymbol{Z}[i] = \text{ReLU}\left(\boldsymbol{W}^{(1)}_{N1} \boldsymbol{h}^{(1)}_{\{i\}} + \boldsymbol{W}^{(1)}_{N2} \boldsymbol{m}^{(2)}_{\mathcal{N}(\{i\})}\right). \tag{3.19}$$

Note the non-probabilistic SAE is more susceptible to overfitting than the SVAE, which is regularized with a KL divergence term. This will be explored further in section 3.5.

3.4 Decoder

Given a set of node-level latents $\boldsymbol{Z} \in \mathbb{R}^{|K_0| \times F}$ generated by the simplicial encoder, the SVAE decoder $p_{\boldsymbol{\theta}} : \mathbb{R}^{|V_0| \times F} \to \text{Dist}(\mathbb{R}^{3 \times 6|K_2|})$ maps the node-level latents to a distribution

over triangles $\boldsymbol{T} \in \mathbb{R}^{3\times 6|K_2|}$. The goal of the decoder is to reconstruct the triangles K_2 present in the original input 2-dimensional abstract simplicial complex K. The decoder takes the form:

$$p_{\boldsymbol{\theta}}(\boldsymbol{T}|\boldsymbol{Z}) = \prod_{\{\,u,v,w\,\}\in\tau(K)} p_{\boldsymbol{\theta}}\left([u,v,w] \in \boldsymbol{T} \mid \boldsymbol{Z}[u], \boldsymbol{Z}[v], \boldsymbol{Z}[w]\right) \tag{3.20}$$

$$= \prod_{\{\,u,v,w\,\}\in\tau(K)} l\left(s(\boldsymbol{Z}[u], \boldsymbol{Z}[v], \boldsymbol{Z}[w])\right), \tag{3.21}$$

where $s : \mathbb{R}^F \times \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}$ is a scoring function, $l : \mathbb{R} \to [0, 1]$ is a non-linear activation, and $\tau(K)$ is the set of possible triangles given the 1-skeleton of simplicial complex K.

Let us explore two possibilities for the scoring function s and non-linear activation l, which define the Bernoulli distribution used to model the existence of triangle $\{\, u, v, w \,\}$. Our first simplicial decoder builds upon the multi-relational decoder DistMult [30], which defines the dot product of two node embeddings $\boldsymbol{Z}[u], \boldsymbol{Z}[v]$ and one relational embedding $\boldsymbol{r}$ as:

$$< \boldsymbol{Z}[u], \boldsymbol{r}, \boldsymbol{Z}[v] >= \sum_{i=1}^{F} \left(\boldsymbol{Z}[u, i]\right)\left(\boldsymbol{r}[i]\right)\left(\boldsymbol{Z}[v, i]\right), \tag{3.22}$$

where $<>$: $\mathbb{R}^F \times \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}$. Note this multi-linear dot product is not an intrinsic geometric distance, unlike the usual dot product over two vectors. This formulation allows us to generalize the VGAE dot product decoder from equation 2.44 to a multi-linear dot product over three vectors. In particular, we define the simplicial decoder of three node

embeddings:

$$p_{\boldsymbol{\theta}}\left([u,v,w] \in \boldsymbol{T} \mid \boldsymbol{Z}[u], \boldsymbol{Z}[v], \boldsymbol{Z}[w]\right) \tag{3.23}$$

$$= \sigma\left(< \boldsymbol{Z}[u], \boldsymbol{Z}[v], \boldsymbol{Z}[w] >\right) \tag{3.24}$$

$$= \sigma\left(\sum_{i=1}^{F} \left(\boldsymbol{Z}[u,i]\right)\left(\boldsymbol{Z}[v,i]\right)\left(\boldsymbol{Z}[w,i]\right)\right), \tag{3.25}$$

where σ is the sigmoid activation function. Note this decoder is only suitable for symmetric relations because this 3-vector dot product is commutative. Although this is a limitation for modeling directed graphs, this decoder is amenable to modeling the undirected relations of abstract simplicial complexes.

Our second simplicial decoder is designed to be analogous to a Čech complex introduced in definition 2.4.2 and takes the form:

$$p_{\boldsymbol{\theta}}\left([u,v,w] \in \boldsymbol{T} \mid \boldsymbol{Z}[u], \boldsymbol{Z}[v], \boldsymbol{Z}[w]\right) \tag{3.26}$$

$$= \left(1 - \tanh\left(\max\left(\|\boldsymbol{Z}[u] - \boldsymbol{Z}[v]\| - r, \|\boldsymbol{Z}[v] - \boldsymbol{Z}[w]\| - r, \|\boldsymbol{Z}[u] - \boldsymbol{Z}[w]\| - r, 0\right)\right)\right). \tag{3.27}$$

Let us unpack this decoder. Remember we defined a Čech complex triangle as the non-empty intersection of three balls of radius r centered at points $P = \{ x, y, z \}$. An equivalent definition is a Čech complex triangle exists if and only if the radius of the minimal ball enclosing points $P = \{ x, y, z \}$ is less than or equal to r. This property is satisfied by P when $\max_{p,q \in P} \|p - q\| \leqslant \sqrt{2} r$ [1]. If we take $P = \{ \boldsymbol{Z}[u], \boldsymbol{Z}[v], \boldsymbol{Z}[w] \}$, then the max term in equation 3.27 is only positive when the distance between a pair of node embeddings in set P exceeds r. In this case, the larger the distance between this pair of nodes, the lower

the probability $[u, v, w] \in \boldsymbol{T}$. In the case where the max term evaluates to 0, all pairs of node embeddings are within distance r and the probability of $[u, v, w] \in \boldsymbol{T} = 1$. Here, $\tanh$ is our choice of activation function because it is a smooth mapping of positive real numbers to [0, 1]. Thus we have constructed a differentiable triangle decoder analogous to the Čech complex. Note radius r is a hyperparameter to the simplicial autoencoder and must be chosen accordingly.

In the non-probabilistic simplicial autoencoder setting, the simplicial dot-product decoder from equation 3.24 and the simplicial Čech decoder from equation 3.27 can be readily used, except the output does not model a Bernoulli distribution and instead corresponds to a soft triangle tensor $\hat{\boldsymbol{T}} \in \mathbb{R}^{3\times|\tau(K)|}$:

$$\hat{\boldsymbol{T}}[u, v, w] = \text{DEC}(\boldsymbol{Z}[u], \boldsymbol{Z}[v], \boldsymbol{Z}[w]),$$

which we will see in section 3.5 can be trained using a binary cross-entropy loss.

3.5 Learning

Similar to the learning objective function of the VGAE in section 2.3.5, the objective function for the SVAE is:

$$\mathcal{L}_\phi = \mathbb{E}_{q_\phi(\boldsymbol{Z}|\boldsymbol{X}_0,\boldsymbol{X}_1,\boldsymbol{X}_2,\boldsymbol{E},\boldsymbol{T})} \left(\log p(\boldsymbol{T} \mid \boldsymbol{Z})\right) - \text{KL}\left(q_\phi((\boldsymbol{Z} \mid \boldsymbol{X}_0, \boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{E}, \boldsymbol{T})) \,||\, p(\boldsymbol{Z})\right). \tag{3.28}$$

Note that distribution p is not parameterized, although a more expressive decoder may be used instead. Triangle tensor $\boldsymbol{T}$ is in coordinate list form, but for convenience let us use

the notation $\boldsymbol{T}[u, v, w] = 1$ to denote $[u, v, w] \in \boldsymbol{T}$ and $\boldsymbol{T}[u, v, w] = 0$ otherwise. The reconstruction term $\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{X}_0,\boldsymbol{X}_1,\boldsymbol{X}_2,\boldsymbol{E},\boldsymbol{T})}\left(\log p(\boldsymbol{T} \mid \boldsymbol{Z})\right)$ is calculated using a binary-cross entropy loss of the following form:

$$\mathcal{L}_{\text{recon}} = -\frac{1}{|\tau(K)|} \sum_{\{\, u,v,w \,\} \in \tau(K)} \boldsymbol{T}[u, v, w] \log l(s(\boldsymbol{Z}[u], \boldsymbol{Z}[v], \boldsymbol{Z}[w])) \tag{3.29}$$

$$+ (1 - \boldsymbol{T}[u, v, w]) \log \left(1 - l(s(\boldsymbol{Z}[u], \boldsymbol{Z}[v], \boldsymbol{Z}[w]))\right). \tag{3.30}$$

Note the use of node-level latents circumvents the problem of simplicial complex isomorphism, where it is prohibitively expensive to determine whether the input abstract simplicial complex K and the output simplicial complex K' are equivalent in structure but with different node labelling. Node-level latents ensures input node u corresponds to its latent embedding and reconstruction, thus enabling a straight-forward calculation of the binary cross-entropy loss. The reparameterization trick and Monte Carlo approximation, introduced in section 2.3.5, are used to estimate the gradient of the SVAE objective function in equation 3.28 with respect to the variational parameters ϕ.

In the non-variational setting, the SAE objective function consists solely of the reconstruction term as there is not need to regularize the deterministic encoder. The reconstruction term is computed with the binary cross-entropy loss in equation 3.29 and the reparametrization trick is irrelevant in this non-stochastic setting.

4

Experiments

The experiments in this chapter evaluate the proposed simplicial variational autoencoder and its non-probabilistic variant on the task of triangle prediction, on synthetic datasets. The purpose of these experiments is to demonstrate functional end-to-end latent variable models that successfully consolidate simplicial complex software with machine learning software.

4.1 Datasets

We generate three synthetic abstract simplicial complex datasets of varying number of triangles. Table 4.1 details the synthetic abstract simplicial complex datasets labelled ASC I, ASC II, and ASC III. The datasets have a varied number of positive triangles and negative triangles. Let $K = K_0 \cup K_1 \cup K_2$ be the 2-dimensional abstract simplicial complex in a given dataset. Here, a positive triangle is defined as a triangle $\{ u, v, w \} \in K_2$ and a negative triangle is defined as a candidate triangle $\{ u, v, w \}$ such that the necessary edges

$\{u, v\}, \{v, w\}, \{u, w\} \in K_1$, but $\{u, v, w\} \notin K_2$.

Each of the datasets contains node-level, edge-level, and triangle-level one-hot input features. These datasets were generated using the library Gudhi [20], where a given number of positive triangles are generated by selecting random triplets of nodes and the corresponding lower-dimensional simplices are added to the simplicial complex. The transformation from a Gudhi simplicial complex into a PyTorch Geometric [7] dataset is our own implementation. These synthetic datasets are small in terms of their number of nodes and serve as simple examples on which to measure the effectiveness of encoding and decoding higher-order structure and features. However, our models are amenable to larger simplicial complexes.

Dataset	Num. Nodes	Num. Edges	Num. +ve Triangles	Num. -ve Triangles
ASC I	52	106	50	10
ASC II	52	126	50	50
ASC III	502	1321	500	640

Table 4.1: Statistics for three synthetic simplicial complex datasets. Each of these datasets is comprised of a 2-dimensional abstract simplicial complex $K = K_0 \cup K_1 \cup K_2$ where $|K_0|$ is equal to Num. Nodes, $|K_1|$ is equal to Num. Edges and $|K_2|$ is equal to Num. +ve Triangles.

4.2 Triangle Prediction for Simplicial Complexes

We evaluate the simplicial variational autoencoder (SVAE) and its non-probabilistic variant (SAE) on the triangle prediction task for the datasets described in section 4.1. The results are presented in table 4.2 where the scores reported are area under the ROC curve (AUC)

and average precision (AP). The receiver operating characteristic (ROC) graphically illustrates the performance of a binary classifier by plotting sensitivity, the rate of true positives, versus 1-specificity, the rate of false positives. The AUC score allows for easy comparison between ROC curves where an AUC of 1.0 is a perfect binary classifier and an AUC of 0.5 is a random binary classifier. Alternately, the AP score captures the trade-off between precision, the ratio of predicted true positives to total predicted positives, and recall, the ratio of predicted true positives to total true positives, of a binary classifier. The AP score:

$$\text{AP} = \sum_{n} (R_n - R_{n-1})\, P_n, \tag{4.1}$$

is a weighted sum of precision scores at each classifier threshold n, where the weight is the gain in recall between subsequent classifier thresholds.

Model	**ASC I**		**ASC II**		**ASC III**	
Encoder-Decoder	AUC	AP	AUC	AP	AUC	AP
GAE-MultDot	0.51 ± 0.01	0.66 ± 0.01	0.66 ± 0.06	0.78 ± 0.05	0.56 ± 0.01	0.69 ± 0.01
VGAE-MultDot	0.41 ± 0.20	0.70 ± 0.10	0.60 ± 0.05	0.72 ± 0.08	0.54 ± 0.02	0.68 ± 0.01
SAE-MultDot	0.49 ± 0.03	0.66 ± 0.01	0.60 ± 0.02	0.64 ± 0.06	$\mathbf{0.56} \pm 0.00$	$\mathbf{0.69} \pm 0.00$
SVAE-MultDot	$\mathbf{0.69} \pm 0.01$	$\mathbf{0.86} \pm 0.01$	$\mathbf{0.68} \pm 0.10$	$\mathbf{0.79} \pm 0.10$	0.54 ± 0.05	0.63 ± 0.03
SAE-Čech	0.50 ± 0.00	0.67 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00
SVAE-Čech	0.50 ± 0.00	0.67 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00

Table 4.2: Experimental results for triangle prediction on three synthetic datasets ASC I, ASC II, and ASC III.

The baselines are VGAE-MultDot and GAE-MultDot; models where the encoder is a VGAE or GAE [14] and the decoder is our multi-linear dot product decoder from equation 3.24. The baseline models treat the input abstract simplicial complex as a graph and output triangle predictions. The simplicial models SVAE-MultDot and SAE-MultDot differ from

the baselines in their choice of encoder; these models utilize simplicial message passing encoders described in section 3.3. The simplicial models SVAE-Čech and SAE-Čech use the Čech decoder from equation 3.27.

All models were run with a similar experimental setup. Each model's encoder is a 2-layer architecture with a latent space of dimension 8 and a hidden layer of dimension 16. Each model was trained for 500 epochs, where the graph autoencoders had a learning rate of 0.01 and the simplicial autoencoders had a learning rate of 0.001. The results reported in table 4.2 are the mean and standard error for 5 runs on fixed dataset splits.

The largest gain in performance over the baselines is by SVAE-MultDot on the dataset ASC I. Note that dataset ASC I has 5 times more positive triangles than negative triangles. SVAE-MultDot is also the best performant model on the dataset ASC II, however, by a smaller margin. For the dataset ASC III, the best model is SAE-MultDot by an even smaller margin. On this dataset, SAE-MultDot performs marginally better than a random model, indicating none of the evaluated models can successfully perform triangle prediction on this dataset.

All models struggle most on the dataset ASC III, which has the largest proportion of negative triangles to positive triangles. It is on datasets with few empty triangles that the simplicial autoencoders perform best. On all abstract simplicial complex datasets, these results corroborate our hypothesis that encoding higher-dimensional simplices using simplicial message passing improves performance on triangle prediction. We expect larger gains would be observed on datasets with meaningful node-level, edge-level, and triangle-level features.

The simplicial autoencoders with a Čech decoder do not perform well for two reasons.

Firstly, the Čech decoder depends on the radius hyperparameter, which cannot adapt to the learned embeddings. Secondly, the Čech decoder has many terms which leads to unstable optimization. The multi-linear dot product performs well, supporting the hypothesis that restricting a decoder's complexity forces the encoder to learn meaningful embeddings [12].

5

Conclusion and Future Work

We have shown that our simplicial autoencoder effectively models higher order relations on the task of triangle prediction, evaluated on synthetic abstract simplicial complex datasets. Our work drew heavily on graph representation learning algorithms and topological data analysis formalism to create a novel simplicial representation learning pipeline; simplicial complex representation using simplex trees; simplicial neural message passing encoders, including variational and deterministic variants; simplicial decoders based on multi-relational graph representation learning and the Čech complex; simplicial autoencoder loss functions.

There are several avenues for future work on simplicial autoencoders. A low hanging fruit is to evaluate the simplicial autoencoders presented in this thesis on datasets with meaningful input features. Co-authorship citation networks, where the task is to determine whether a triplet of authors have co-authored a paper, is of particular interest. For this, hypergraph relation prediction models can serve as useful baselines. Another direction is to remedy the Čech decoder so that the radius is a learnable parameter. Using sophisticated

hyperparameter sweep techniques to determine the best fixed radius is not enough; the decoder radius must change to accommodate the geometry of the learned latent embeddings. Finally, a broader research direction is to design a new encoder, perhaps a new convolutional network for simplicial complexes, capable of capturing topological properties other than local structure.

Bibliography

[1] ATTALI, D., LIEUTIER, A., AND SALINAS, D. Vietoris-Rips Complexes also Provide Topologically Correct Reconstructions of Sampled Shapes. *Computational Geometry 46*, 4 (May 2012), 448–465. special issue 27th Annual Symposium on Computational Geometry (SoCG 2011).

[2] BENGIO, Y. Learning deep architectures for ai. *Foundations 2* (01 2009), 1–55.

[3] BENSON, A. R., ABEBE, R., SCHAUB, M. T., JADBABAIE, A., AND KLEINBERG, J. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences 115*, 48 (2018), E11221–E11230.

[4] BOISSONNAT, J.-D., AND MARIA, C. The simplex tree: An efficient data structure for general simplicial complexes. In *Algorithms – ESA 2012* (Berlin, Heidelberg, 2012), L. Epstein and P. Ferragina, Eds., Springer Berlin Heidelberg, pp. 731–742.

[5] CARLSSON, G., ZOMORDIAN, A., COLLINS, A., AND GUIBAS, L. J. Persistence barcodes for shapes. *International Journal of Shape Modeling 11*, 02 (2005), 149–187.

[6] DEFFERRARD, M., BRESSON, X., AND VANDERGHEYNST, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural*

Information Processing Systems (2016), D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., pp. 3844–3852.

[7] FEY, M., AND LENSSEN, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428* (2019).

[8] GALLEGO, J. Simplicial autoencoders. Master's thesis, 2018.

[9] GHRIST, R. Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society 45*, 1 (2008), 61–75.

[10] GHRIST, R. W. *Elementary applied topology*, vol. 1. Createspace Seattle, 2014.

[11] GILMER, J., SCHOENHOLZ, S. S., RILEY, P. F., VINYALS, O., AND DAHL, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (2017), ICML'17, JMLR.org, p. 1263–1272.

[12] HAMILTON, W. L. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning 14*, 3, 1–159.

[13] KINGMA, D. P., AND WELLING, M. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (2014).

[14] KIPF, T. N., AND WELLING, M. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).

[15] KIPF, T. N., AND WELLING, M. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations* (2017), ICLR '17.

[16] KORMAN, E. O. Autoencoding topology. *arXiv preprint arXiv:1803.00156* (2018).

[17] KULLBACK, S., AND LEIBLER, R. A. On information and sufficiency. *The annals of mathematical statistics 22*, 1 (1951), 79–86.

[18] KUMAR, T., DARWIN, K., PARTHASARATHY, S., AND RAVINDRAN, B. HPRA: hyperedge prediction using resource allocation. *CoRR abs/2006.11070* (2020).

[19] LECUN, Y., BENGIO, Y., AND HINTON, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[20] MARIA, C., BOISSONNAT, J.-D., GLISSE, M., AND YVINEC, M. The gudhi library: Simplicial complexes and persistent homology. In *Mathematical Software – ICMS 2014* (Berlin, Heidelberg, 2014), H. Hong and C. Yap, Eds., Springer Berlin Heidelberg, pp. 167–174.

[21] MOOR, M., HORN, M., RIECK, B., AND BORGWARDT, K. Topological autoencoders. In *International Conference on Machine Learning* (2020), PMLR, pp. 7045–7054.

[22] NICKEL, M., MURPHY, K., TRESP, V., AND GABRILOVICH, E. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE 104*, 1 (2015), 11–33.

[23] PASZKE, A., GROSS, S., CHINTALA, S., CHANAN, G., YANG, E., DEVITO, Z., LIN, Z., DESMAISON, A., ANTIGA, L., AND LERER, A. Automatic differentiation in pytorch. In *NIPS-W* (2017).

[24] REZENDE, D. J., MOHAMED, S., AND WIERSTRA, D. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning* (Bejing, China, 22–24 Jun 2014), E. P. Xing and T. Jebara, Eds., vol. 32 of *Proceedings of Machine Learning Research*, PMLR, pp. 1278–1286.

[25] ROSENBLATT, F. Principles of neurodynamics: Perceptions and the theory of brain mechanisms.

[26] SCARSELLI, F., GORI, M., TSOI, A. C., HAGENBUCHNER, M., AND MONFARDINI, G. The graph neural network model. *IEEE Transactions on Neural Networks 20*, 1 (2009), 61–80.

[27] SIMONOVSKY, M., AND KOMODAKIS, N. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks* (2018), Springer, pp. 412–422.

[28] WERBOS, P. J. Applications of advances in nonlinear sensitivity analysis. In *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC* (1981), pp. 762–770.

[29] YADATI, N., NIMISHAKAVI, M., YADAV, P., NITIN, V., LOUIS, A., AND TALUKDAR, P. Hypergcn: A new method for training graph convolutional networks on hypergraphs. In *Advances in Neural Information Processing Systems* (2019), H. Wal-

lach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc.

[30] YANG, B., YIH, W.-T., HE, X., GAO, J., AND DENG, L. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575* (2014).

[31] ZITNIK, M., AGRAWAL, M., AND LESKOVEC, J. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics 34*, 13 (06 2018), i457–i466.

Acronyms

COO	Coordinate list
CNN	Convolutional neural network
GAE	Graph autoencoder
GCN	Graph convolutional network
GNN	Graph neural network
MLP	Multilayer perceptron
MPNN	Message passing neural network
SNN	Simplicial neural network
SAE	Simplicial autoencoder
SVAE	Simplicial variational autoencoder
VAE	Variational autoencoder
VGAE	Variational graph autoencoder