# Incidence Networks For Geometric Deep Learning

by

Marjan Albooyeh

B.Sc., Amirkabir University of Technology, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

**Master of Science**

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

October 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

## Incidence Networks For Geometric Deep Learning

submitted by **Marjan Albooyeh** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

**Examining Committee:**

Leonid Sigal, Computer Science
*Supervisor*

Mark Schmidt, Computer Science
*Second Reader*

# Abstract

Sparse incidence tensors can represent a variety of structured data. For example, we may represent attributed graphs using their node-node, node-edge, or edge-edge incidence matrices. In higher dimensions, incidence tensors can represent simplicial complexes and polytopes. In this work, we formalize incidence tensors, analyze their structure, and present the family of equivariant networks that operate on them. We show that any incidence tensor decomposes into invariant subsets. This decomposition, in turn, leads to a decomposition of the corresponding equivariant layer that allows efficient and intuitive pooling-and-broadcasting implementation, for both dense and sparse tensors. We demonstrate the effectiveness of this family of networks by reporting state-of-the-art on graph learning tasks for many targets in the QM9 dataset.

# Lay Summary

In this work, we introduced Incidence Networks as a general approach for learning permutation equivariant neural layers operating on data structures that can be encoded using their incidence representation. We analyzed the incidence tensors and the symmetry groups acting on them. An efficient implementation of the equivariant layer is proposed using a novel interpretation of pooling and broadcasting operations.

We evaluated our model on a large dataset of molecules represented as graphs, where the task is to estimate the chemical properties of the molecules. We achieved state-of-the-art results that also support our theoretical analysis.

# Preface

The work presented in this thesis was performed in collaboration with Daienle Bertolini, and my former supervisor Siamak Ravanbakhsh. Writing the code and running all the experiments was done by me with the help of Daniele and Siamak.

A version of this work has been submitted to the International Conference on Artificial Intelligence and Statistics (AISTATS 2020) and is under anonymous review at the moment of thesis submission.

# Table of Contents

# List of Tables

# List of Figures

# Acknowledgments

I would like to express my deepest gratitude to my supervisor, Prof. Siamak Ravanbakhsh, for his continuous support and guidance throughout my program. He patiently provided me valuable feedback and suggestions whenever I needed help. I will always be grateful for all that I have learned from him.

I also would like to thank Daniele Bertolini, our insightful colleague, for his contribution and for many helpful conversations and suggestions. It was a great opportunity for me to work with him in this project.

Also, many thanks to Prof. Leonid Sigal and Prof. Mark Schmidt for taking time to read this work and for their constructive feedback.

Last but not least, to my parents and my sister, thank you for your endless love and support and for encouraging me in all of my pursuits and inspiring me to follow my dreams.

*For Maryam, my lovely sister*

# Chapter 1

# Introduction

Many interesting data structures have alternative tensor representations. For example, we can represent a graph using both node-node and node-edge sparse incidence matrices. We can extend this incidence representation to work with signals on simplicial complexes and polytopes of arbitrary dimension, such as mesh, polygons, and polyhedra. Our goal is to design deep models for these structures.

In our formalism, an incidence tensor is a tensor with specific sparsity pattern that models the incidence of faces of a geometric object. For example, rows and columns in a node-edge incidence matrix are indexed by faces of dimensions zero (nodes) and one (edges). Moreover each edge (column) is incident to exactly two nodes (rows). Indeed this sparsity pattern has important information about the underlying geometric structure. This is because sparsity preserving permutations often match the automorphism group (a.k.a. symmetry group) of the geometric object, and one may design neural layers that are equivariant/invariant with respect to these symmetries.

Study of these symmetries also leads to a decomposition of the tensor. In particular, we observe that any incidence tensor, under the action of its symmetry group, is isomorphic to the disjoint union of *face-vectors*, such as node and edge vectors in a graph. This decomposition also breaks up the design of equivariant maps for arbitrary incidence tensors into design of such maps for face-vectors.

Another factor in the design of equivariant layers is the choice of the symmetry group: in addition to the automorphism group, we consider symmetric group, product of symmetric groups, and discuss the implications of each choice. In particular, these choices are affected by the sparsity of representation and the variability of structure across data-points (e.g., modeling the data on one versus many graphs). We also provide an extensive experimental evaluation of incidence networks on one of the largest graph datasets (QM9). The results support our theoretical findings and establish a new state-of-the-art for several targets.

The rest of this thesis is organized as follows: In Chapter 2, we review the related works and literature. Chapter 3 is dedicated to the detailed definition of incidence tensors followed by a few examples to represent different geometric structures. In the second part of this chapter, we discuss the symmetry groups and the decomposition of incidence tensors. In Chapter 4, we describe the properties of the equivariant maps for such tensors. Chapter 5 includes our experimental results along with the details of the dataset and training procedure. Finally, in Chapter 6 we summarize our main contributions and

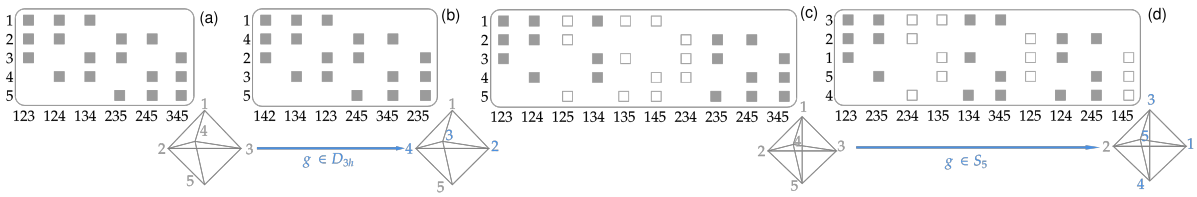



**Figure 1.1: a**) The sparsity pattern in the node-face incidence matrix for an (undirected) *triangular bi-pyramid* (concatenation of two tetrahydra). Note that each face (column) is adjacent to exactly three nodes. **b**) Nodes are permuted using a member of the symmetry group of the object $\pi \in \mathcal{D}_{3h} \leq \mathcal{S}_5$. This permutation of nodes imposes a natural permutation action on the faces in which $\{\delta_1, \delta_2, \delta_3\} \mapsto \{\pi \cdot \delta_1, \pi \cdot \delta_2, \pi \cdot \delta_3\}$. Note that permutations from the automorphism group preserve the sparsity pattern of the incidence matrix. **c**) The geometric object of (a) after *densification*: the incidence matrix now includes all possible faces of size three, however, it still maintains a specific sparsity pattern. **d**) After densifying the structure, *any* permutation of nodes, (and corresponding permutation action on faces of the dense incidence matrix) preserves its sparsity pattern.

discuss possible future directions. Additional results are provided in the Appendix.

# Chapter 2

# Related Works

In this chapter, we briefly review some of the closely related works in graph learning and equivariant deep learning.

## 2.1 Graph Learning

Recently, Deep Learning models e.g., Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and AutoEncoders have achieved great performance in a broad range of Machine Learning tasks from object detection and speech recognition, to machine translation. These models have been particularly successful when the data is defined on Euclidean (grid) space such as images, text or signals. However, in many real-world problems the data is defined on non-Euclidean space in form of graphs, meshes or manifolds. Deep Learning with these types of structured data is a very active area of research; see [2, 13]. Graph structures, in particular, have received a lot of attention due to their potential to model the interactions and dependencies between many different objects in various domains, such as: social networks, molecular graphs, recommender systems, brain connectome, knowledge graphs, etc.

### 2.1.1 Graph Neural Networks

Motivated by the success of deep learning in different fields such as computer vision, natural language processing, and signal processing , Graph Neural Networks (GNNs) are introduced as machine learning models that can directly deal with complex graph structures. These networks are mostly used to learn informative representations of nodes and edges using graph local neighborhood structures, which later can be used in different tasks such as node classification, link prediction, graph classification, etc.

The idea of graph neural networks goes back to the work of Scarselli et al. [33], who introduced a general iterative framework for learning neural functions on graphs. In this approach, nodes' latent representations are recurrently updated, using the information received by the neighbors. This process will be continued until a stable equilibrium is reached. Despite the inital success of these models, they usually suffer from heavy local computations and large time complexity.

### 2.1.2 Message Passing Neural Networks

The next generation of the GNNs mostly fall into the category of Message Passing Neural Networks (MPNNs), where the information is passing through messages in the graph. Gilmer et al. [11] developed different variations of MPNNs to learn iterative updates of messages that are passed among nodes in a graph. Assuming all the nodes are initialized with a hidden state, in each message passing iteration, a message for a specific node is an aggregation of all the messages coming from its neighbors. These messages are calculated using a neural function that is shared among all the nodes in the graph. After the message passing phase, each node's hidden state will be updated using a learnable update function given the calculated message and the current hidden state. After a certain amount of iterations, the final hidden state of each node, will be considered as the high level node representation. The difference between models in this category mostly comes from the way the message and update functions are defined.

Duvenaud et al. [10] used a simple concatenation of the neighbor's hidden state and the corresponding edge feature to find a message between two adjacent nodes. The node update function is simply defined as the projection of the message using a learnable matrix which is different based on the node's degree. Li et al. [26] proposed Gated Graph Neural Network (GGNN) that projects neighbor's hidden state with a learnable matrix which depends on the type of the edge between two nodes. After updating all the messages, a Gated Recurrent Unit (GRU) is employed to update each node's hidden state based on the calculate message and previous hidden state. The MPNN framework generalizes several other graph neural architectures [18, 34], including the spectral methods that we discuss later.

### 2.1.3 Convolutional Graph Neural Networks

Convolutional Graph Neural Networks (ConvGNNs) are another line of methods that apply the convolution operation to graphs. The original ConvGNNs introduced based on a signal processing perspective, where the spectral convolution operation is defined using graph Fourier transform [3, 4]. This operation projects the input graph signal with eingenvectors of the normalized graph Laplacian and a learnable filter. Therefore, the convolution filter depends on the eigen-decomposition of the Laplacian matrix, which is computationally expensive. While principled, in its complete form, the Fourier bases extracted from the Laplacian are instance dependent and lack of any parameter or function sharing across the graph limits their generalization.

Most of the follow-up works proposed a single-parameter simplification of spectral method that addresses these limitations. Chebyshev Spectral CNN (ChebNet) [9] and CayleyNet [25] used k-order Chebyshev polynomials of the Lalacian and Cayley polynomials to approximate the filter respectively. Later, Kipf and Welling [21] introduced Graph Convolution Network (GCN) as a first order approximation of the ChebNet. Interestingly this minimal graph convolution model can also be derived from neural message passing perspective, where the message function is parametrized by the eigenvectors of the laplacian and the learned parameters of the model [11].

## 2.2 Equivariant Layer

Equivariance constrains the predictions of a model $\phi : \mathbb{X} \mapsto \mathbb{Y}$ under a group $\mathcal{G}$ of transformations of the input, such that

$$\phi(\pi \cdot x) = \pi \cdot \phi(x), \forall x \in \mathbb{X}, \forall \pi \in \mathcal{G}. \tag{2.1}$$

Here $\pi \cdot x$ is a "consistently" defined transformation of $x$ parameterized by $\pi \in \mathcal{G}$, while $\pi \cdot \phi(x)$ denotes the corresponding transformation of the output. For example, in a convolution layer [24], $\mathcal{G}$ is the group of discrete translations, and Eq. (2.1) means that any translation of the input leads to the same translation of the output. When $\phi : x \mapsto \sigma(\Lambda x)$ is a standard feed-forward layer with parameter matrix $\Lambda$, and $\mathcal{G}$ has a permutation action, the equivariance property of Eq. (2.1) enforces all elements of $\Lambda$ that form an orbit under the diagonal action of $\mathcal{G}$ to get tied together [32]; see also [36, 37]. Due to the symmetry of $\Lambda$, for homogeneous spaces, the linear operation $\Lambda x$ can also be interpreted as convolution [6, 7, 22].

## 2.3 Equivariant Deep Learning

Zaheer et al. [40] proposed DeepSets, a permutation invariant/equivariant architecture for inputs represented in the form of sets. In this architecture, each layer follows a simple parameter-sharing scheme: one parameter is used to project all the set elements to the output and another parameter for projecting the result of pooling the input across set-members and broadcast it back to the set initial dimension.

Hartford et al. [14] employed permutation equivariant layer to model the interactions among two or more sets. A practical application of this model is in user-movie rating systems, where the data can be represented in a form of an exchangeable matrix such that any row/column wise permutation preserves the encoding of the input. They showed that the parameter tying in this case results in 4 different parameters shared among elements of the parameter matrix where each of them corresponds to a particular pooling/broadcasting operation over the rows/cols of the input. Although this model is related incidence tensors that model interaction across faces, the main distinction here is that the group action on these sets is not independent as it is assumed in their work; see Section 4.4.

In case of graphs, it is clear that the symmetries are defined in form of node permutations that preserve the graph structure. Learning GNNs that are invariant/equivariant to such permutations has attracted considerable research attention recently. The goal is to design a permutation equivariant model such that the output is permuted when the input graph is permuted. This is particularly helpful when performing node-level tasks on graphs. For graph-level tasks, the GNN must be permutation invaraint which means the output must not depend on the order of the nodes in graph.

Maron et al. [27] proposed characteristics of a permutation invariant and equivariant linear layer in graph networks. They showed that the parameter space in such layer is independent of the size of input and it only depends on the order of input/output tensors. In this framework, a permutation equivariant layer is formed as linear combination of 15 basic operators on a graph and a permutation invariant layer can be formalized using 4 operators. See [27] for more details.

For general incidence tensors, the permutation actions are dependent yet not identical. These equivariant layers for interactions within and between sets are further generalized to multiple types of

interactions in [12]. Several recent works investigate the universality of such equivariant networks[5, 19, 29]. A flexible approach to equivariant and geometric deep learning where a global symmetry is lacking is proposed in [8].

# Chapter 3

# Incidence Tensors

This chapter first defines an *incidence tensor*, then gives several examples, showing how it can represent geometric objects such as a graphs, polytopes or simplicial complexes. Later, in Section 3.2, we discuss the symmetry groups associated with incidence tensors and the decomposition caused by the action of those groups on the incidence tensors.

## 3.1 Incidence Tensor Representation

Let $[N] = \{1, \ldots, N\}$ denote a set of nodes. A *directed face* of size $M$ is an ordered tuple of $M$ distinct nodes $\delta \in [N]^M \mid \delta_i \neq \delta_j \forall i \neq j$. Following a similar logic an *undirected face* $\delta \subseteq [N]$, is a subset of $[N]$ of size $M$. We use $\delta^{(M)}$ when identifying the size of the face – i.e., $|\delta^{(M)}| = M$.

**Definition 1** (Incidence Tensor)**.** *An* incidence tensor $\mathbf{X}_{\delta_1,\ldots,\delta_D|\Sigma}$ *is an order D tensor, where each dimension d is indexed by faces of size* $M_d$ $\delta_d = \{\delta_{d,1}, \ldots, \delta_{d,M_d}\}$. $\Sigma = \{\bar{\delta}_1, \ldots, \bar{\delta}_C\}$ *identifies the sparsity structure of* $\mathbf{X}$ *as follows: all the indices* $\bar{\delta}_{m_1} \ldots, \bar{\delta}_{m_c} \in \bar{\delta}_c$, $\forall \bar{\delta}_c \in \Sigma$ *should be equal for any non-zero entry of* $\mathbf{X}$.

Note that in contrast to $\delta$, $\bar{\delta}$ does not identify a face, but simply constrains node indices of different faces to be equal, and in doing so it identifies the incidence structure of faces.

This formalism can represent a variety of different geometric structure as demonstrated in the following sections.

### 3.1.1 Simplicial Complexes

Before discussing general simplicial complexes let us consider the important example of undirected graphs.

### Undirected Graph

I) The *node-node* incidence matrix $\mathbf{X}_{\{\delta_1\},\{\delta_2\}}$ of an undirected graph is indexed by a pair of nodes, and there are no sparsity constraints. II) The *node-edge* incidence matrix is denoted by $\mathbf{X}_{\{\delta_1\},\{\delta_2,\delta_3\}|\{\{\delta_1,\delta_2\}\}}$. This matrix is indexed by nodes $\{\delta_1\}$ and edges $\{\delta_2,\delta_3\}$. The entries can be non-zero only when $\delta_1 = \delta_2$, meaning that the edge $\{\delta_1,\delta_3\}$ is adjacent to the node $\{\delta_1\}$. An alternative notation for the same incidence matrix is $\mathbf{X}_{\{\delta_1\},\{\delta_1,\delta_2\}}$ (notice the repeated $\delta_1$ index). III) The *edge* incidence vector $\mathbf{X}_{\{\delta_1,\delta_2\}}$ also has no sparsity constraints. IV) Finally, the element $\mathbf{X}_{\{\delta_1,\delta_2\},\{\delta_1,\delta_3\}}$ of the *edge-edge* incidence matrix is non-zero wherever two edges are incident.

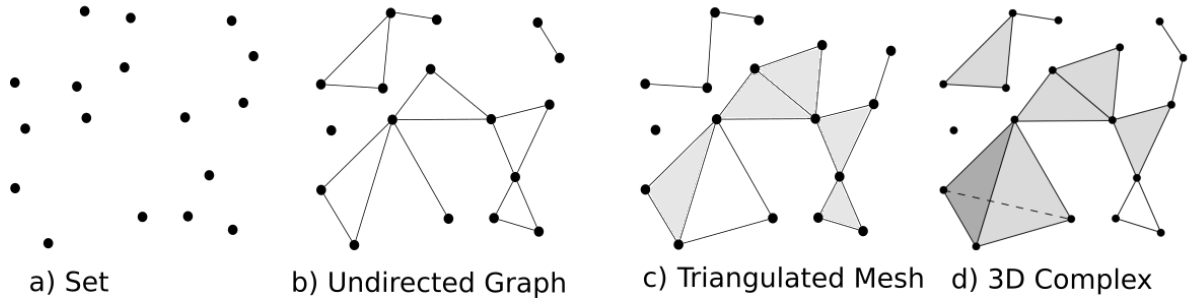
### Motifs Tensors

One may also consider *higher order faces* corresponding to particular *motifs* – e.g., in the edge-edge-edge incidence indexed by $\{(\delta_1,\delta_2),(\delta_2,\delta_3),(\delta_3,\delta_4)\}$, the non-zero elements correspond to paths of length 3. As another example, $\{(\delta_1,\delta_2),(\delta_2,\delta_3),(\delta_3,\delta_1)\}$ the non-zero elements correspond to directed triangles.

### Undirected Simplicial Complex

An *abstract simplicial complex* $\Delta \subseteq 2^{[N]}$ is a collection of faces, closed under the operation of taking subsets – that is $(\delta_1 \in \Delta \text{ and } \delta_2 \subset \delta_1) \Rightarrow \delta_2 \in \Delta$. Each $\delta \in \Delta$ is a *face* of *dimension* $|\delta|-1$. Zero-dimensional faces are called *vertices*, and maximal faces are called *facets*. The dimension of $\Delta$ is the dimension of its largest facet. Each dimension of an incidence tensor may be indexed by faces of specific dimension. Two undirected faces of different dimension $\delta,\delta' \in \Delta$ are incident if one is a subset of the other. This type of relationship as well as alternative definitions of incidence between faces of the *same* dimension can be easily accommodated in the form of equality constraints in $\Sigma$.

**Example 1.** *A zero dimensional simplicial complex is a* set *of points that we may represent using an incidence vector. At dimension one, we get* undirected graphs*, where faces of dimension one are the edges.* Triangulated mesh *is an example of two-dimensional simplicial complex; see figure below.*




*The* triangular bi-pyramid *of Fig. 1.1 is an example of 3 dimensional simplicial complex with 5 nodes, 9 edges, 7 two-dimensional faces, and two three-dimensional faces. The node-face incidence matrix in Fig. 1.1(a) is expressed by* $\mathbf{X}_{\{\delta_1\},\{\delta_1,\delta_2,\delta_3\}}$ *in our formalism.*

**Directed Faces**

Although not widely used, a *directed* simplicial complex can be defined similarly. The main difference is that faces are *sequences* of the nodes, and $\Delta$ is closed under the operation of taking a subsequence. As one might expect, the incidence tensor for directed simplicial complexes can be built using *directed faces* in our notation.

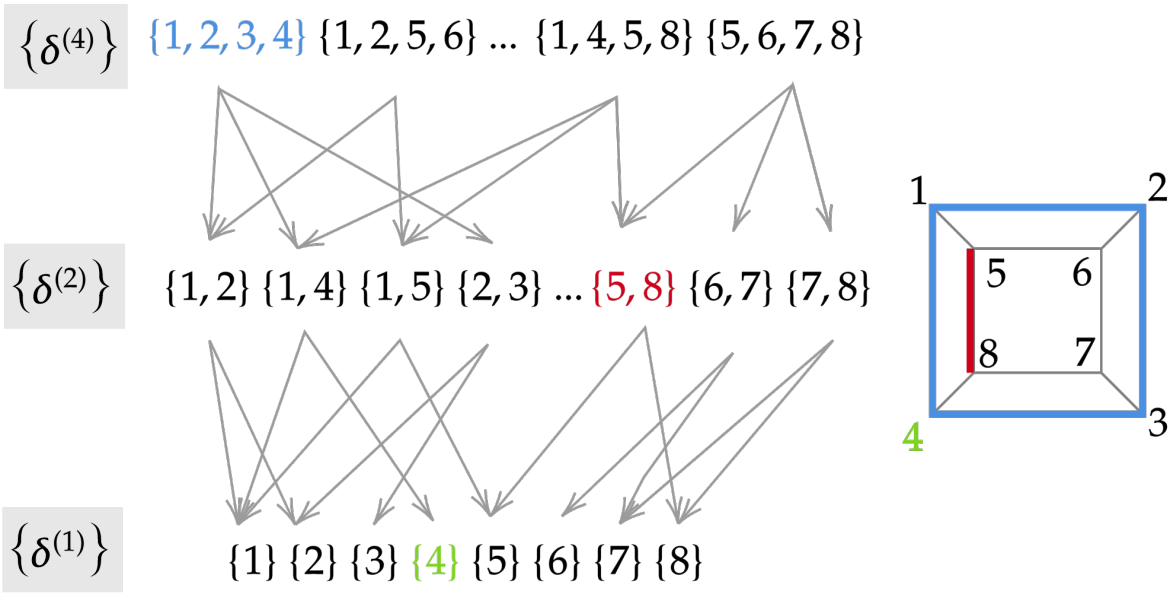



**Figure 3.1:** Representation of a cube as a (graded) partially ordered set. The incidence structure of the poset as well as face attributes are encoded in the incidence matrix.

### 3.1.2 Polygons, Polyhedra, and Polytopes

A *polytope* is a generalization of polygone and polyhedron to higher dimensions. The structure of an *abstract polytope* is encoded using a *graded* partially ordered set (poset). A poset is a set equipped with a partial order that enables transitive comparison of certain members. A poset $\Pi$ is graded if there exists a rank function $\text{rank} : \Pi \to N$ satisfying the following constraints: $\delta < \delta' \Rightarrow \text{rank}(\delta) < \text{rank}(\delta') \quad \forall \delta, \delta' \in \Pi$ $\nexists\, \delta'' \in \Pi \; s.t. \; \delta < \delta'' < \delta' \Rightarrow \text{rank}(\delta') = \text{rank}(\delta) + 1$ An abstract polytope is a set $\Pi$ of partially ordered faces of different dimension. In a geometric realisation, the partial order is naturally defined by the inclusion of a lower-dimensional face in a higher dimensional one (e.g., an edge that appears in a face of a cube). Fig. 3.1 shows the partial order for a cube, where we continue to use a set of nodes to identify a face.

We can define the incidence structure similar to simplicial complexes. For example, we may assume that two faces $\delta, \delta' \in \Pi$ of different dimension (rank) are incident iff $\delta < \delta'$, or $\delta > \delta'$. Similarly, we may assume that two faces $\delta, \delta'$ of the same dimension $d$ are incident iff there is a face $\delta''$ of dimension $d-1$ incident to both of them $\delta'' < \delta, \delta'$.[1]

[1] Note that if the polytope is *irregular*, faces of similar rank may have different sizes – e.g., consider the soccer ball where

## 3.2 Symmetry & Decomposition

The automorphism group $\mathcal{A}ut(\mathbf{X}) \leq \mathcal{S}_N$ associated with an incidence tensor is the set of all permutations of nodes that maps every face to another face, and therefore preserve the sparsity ($\mathbf{X}_{\pi\cdot\delta_1,\ldots,\pi\cdot\delta_D} \neq 0 \Leftrightarrow \mathbf{X}_{\delta_1,\ldots,\delta_D} \neq 0$) $\Leftrightarrow \pi \in \mathcal{A}ut(\mathbf{X})$ where the action of $\mathcal{A}ut(\mathbf{X})$ on the faces is naturally defined as

$$\pi \cdot (\delta_1, \ldots, \delta_M) = (\pi \cdot \delta_1, \ldots, \pi \cdot \delta_M). \tag{3.1}$$

See Fig. 1.1(a,b) for an example. We may then construct $\mathcal{A}ut(\mathbf{X})$-equivariant linear layers through parameter-sharing. However, the constraints on this linear operator varies if our dataset has incidence tensors with different sparsity patterns. For example, a directed graph dataset may contain a fully connect graph with automorphism group $\mathcal{S}_N$ and a cyclic graph with automorphism group $\mathcal{C}_N$. For these two graphs, node-node and node-edge incidence matrices are invariant to the corresponding automorphism groups, necessitating different constraints on their linear layer; indeed in the former case we get the equivariant layer of DeepSet [40], and in the latter case we recover 1D (circular) convolution. To remedy the problem with model-sharing across instances, we *densify* all incidence tensors so that all directed or undirected faces of a given dimension are present. Now, one may use the same automorphism group $\mathcal{S}_N$ across all instances; see Fig. 1.1(c,d).

Next, we consider the incidence tensor as a G-set, and identify the orbits of $\mathcal{S}_N$ action. These orbits remain invariant under the action of the original automorphism group as well, and therefore our decomposition remains useful in application of incidence networks using the automorphism group of the sparse structure.

**Theorem 3.2.1.** *The action of $\mathcal{S}_N$ on any incidence tensor $\mathbf{X}$ decomposes into orbits that are each isomorphic to a face-vector:*

$$\{\delta_1, \ldots, \delta_D \,|\, \Sigma\} \;\cong\; \bigcup_m \kappa_m \{\delta^{(m)}\}, \tag{3.2}$$

*where $\kappa_m$ is the* multiplicity *of faces of size m. The value of $\kappa_m$ is equal to the number of partitioning of the set of all indices $\{\delta_{1,1}, \ldots, \delta_{1,M_1}, \ldots, \delta_{D,M_D}\}$ into m non-empty partitions, such that $\delta_{d,m}\, \forall m \in [M_d]$ belong to different partitions, and members of $\bar{\delta} \in \Sigma$ belong to the same partition.*

*Proof.* Consider the incidence tensor $\mathbf{X}_{\delta_1,\ldots,\delta_D|\Sigma}$. Assume there exist two node indices $\delta_{d,m}$, $\delta_{d',m'}$ within two face indices $\delta_d$ and $\delta_{d'}, d' \neq d$ that are not constrained to be equal by $\Sigma$; therefore they can be either equal or different. The action of $\mathcal{S}_N$ (and any of its subgroups) maintains this (lack of) equality, that is

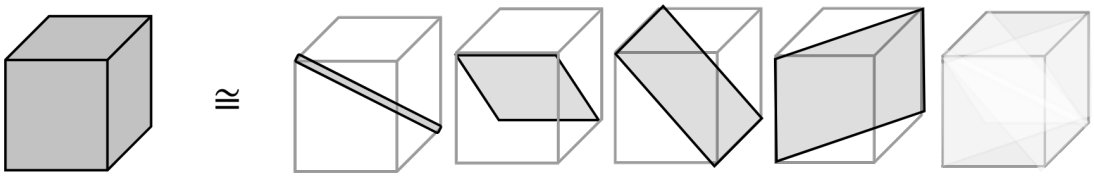
$$\pi \cdot \delta_{d,m} = \pi \cdot \delta_{d',m'} \Leftrightarrow \delta_{d,m} = \delta_{d',m'} \forall \pi \in \mathcal{S}_N. \tag{3.3}$$

pentagons and hexagons have the same rank. Although this irregularity does not undermine the ideas around densification and group action on the incidence tensor, in the following sections, for simplicity, we continue to assume all faces indexing the same dimension of the incidence tensor have the same size.

This means that the index set $\{\delta_1, \ldots, \delta_D \,|\, \Sigma\}$, can be partitioned into two disjoint G-sets $\{\delta_1, \ldots, \delta_D \,|\, \Sigma\} = \{\delta_1, \ldots, \delta_D \,|\, \Sigma, \delta_{d,m} = \delta_{d',m'}\} \uplus$
$\{\delta_1, \ldots, \delta_D \,|\, \Sigma, \delta_{d,m} \neq \delta_{d',m'}\}$, with $\delta_{d,m} = \delta_{d',m'}$ in one G-set and $\delta_{d,m} \neq \delta_{d',m'}$ in the other. We may repeat this partitioning recursively for each of the resulting index-sets. This process terminates with *homogeneous* G-sets where any two indices $\delta_{d,m}$ and $\delta_{d',m'}$ are either constrained to be equal or different. It follows that if we aggregate all equal indices, we are left with a set of indices that are constrained to be different, and therefore define a face $\delta^{(m)}$.

The number of ways in which we are left with $m$ "different" indices $\kappa_m$ is given by the number of partitions of $\{\delta_{1,1}, \ldots, \delta_{1,M_1}, \ldots, \delta_{D,M_D}\}$ into $m$ non-empty sets, where two elements in the same partition are constrained to be equal. This partitioning is further constrained by the fact that elements within the same face $\delta_d$, are constrained to be different and therefore should belong to different partitions. Moreover, $\Sigma$ adds its equality constraints. □

**Example 2** (Node-adjacency tensors)**.** *Consider an order D node-node-...-node incidence tensor* $\mathbf{X}_{\{\delta_1\},\ldots\{\delta_D\}}$ *with no sparsity constraints. In this case, the multiplicity* $\kappa_m$ *of Eq.* (3.2) *corresponds to the number of ways of partitioning a set of D elements into m non-empty subsets and it is also known as Stirling number of the second kind (written as* $\left\{{D \atop m}\right\}$*). Each partition of size m identifies a face-vector* $\mathbf{X}_{\delta^{(m)}}$ *for a face of size m. These faces can be identified as hyper-diagonals of order m in the original adjacency tensor* $\mathbf{X}$*. For example, as shown in the figure below,* $\mathbf{X}_{\{\delta_1\},\{\delta_2\},\{\delta_3\}}$ *decomposes into a node-vector (the main diagonal of the adjacency cube), three edge-vectors (isomorphic to the three diagonal planes of the cube adjacency, with the main diagonal removed), and one hyper-edge-vector (isomorphic to the adjacency cube, where the main diagonal and diagonal planes have been removed). Here,* $\kappa_1 = \left\{{3 \atop 1}\right\} = 1$*,* $\kappa_2 = \left\{{3 \atop 2}\right\} = 3$*, and* $\kappa_3 = \left\{{3 \atop 3}\right\} = 1$*.*



# Chapter 4

# Equivariant Maps for Incidence Tensors

As shown in the previous chapter, any incidence tensor can be decomposed into disjoint union of face-vectors, that are invariant sets under the action of symmetry group. An implication is that any equivariant map from an incidence tensor to another also decomposes into equivariant maps between face-vectors.

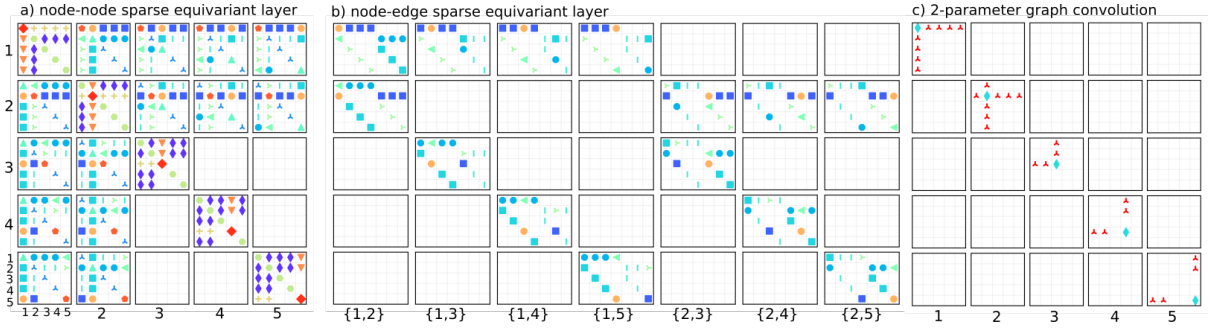



**Figure 4.1:** Parameter-sharing in the receptive field of equivariant map for (**left**) node-node and (**middle**) node-edge incidence matrix, with sparse input and outputs, and (**right**) for comparison the parameter-sharing in the commonly used two-parameter graph convolution.
**BLOCK STRUCTURE.** The adjacency structure of the undirected graph with 5 nodes and 7 edges is evident from the sparsity patterns. Here, each inner block shows the parameter-sharing in the receptive field of the corresponding output unit. For example, the block on row 1 and column $\{1,3\}$ of the (middle) figure shows the dependency of the output incidence matrix at that location on the entire input incidence matrix.
**DECOMPOSITION.** The total number of unique parameters in (left) is 15 compared to 7 for the (middle). As shown in Section 4.2 the 15 ($=7+2+3+3$) parameter model decomposes into 4 linear maps, one of which is isomorphic to the 7 parameter model. One could also identify the 7 unique symbols of (middle) in the parameter-sharing of the (left). Note that these symbols appear on the off-diagonal blocks and off-diagonal elements within blocks, corresponding to input and output edges.

Let $\Lambda^{M\to M'}$ be a linear function (here represented as a tensor) that maps a face-tensor of order $M$ to a

face-tensor of order $M'$,

$$\Lambda^{M\to M'} : R^{\overbrace{N\times N\times\cdots\times N}^{M}} \to R^{\overbrace{N\times N\times\cdots\times N}^{M'}} \tag{4.1}$$

$$\mathbf{X}_{\delta_1,\ldots,\delta_M} \to \mathbf{X}'_{\delta'_1\ldots\delta'_{M'}} = \Lambda^{\delta_1,\ldots,\delta_M}_{\delta'_1,\ldots,\delta'_{M'}} \mathbf{X}_{\delta_1,\ldots,\delta_M}, \tag{4.2}$$

where repeated indices are summed over. Equivariance to $\mathcal{S}_N$ is realized through a symmetry constraint on $\Lambda$,

$$\Lambda^{\pi\cdot\delta_1,\ldots,\pi\cdot\delta_M}_{\pi\cdot\delta'_1\ldots\pi\cdot\delta'_{M'}} = \Lambda^{\delta_1\ldots\delta_M}_{\delta'_1\ldots\delta'_{M'}} \quad \forall\, \pi \in \mathcal{S}_N, \tag{4.3}$$

which ties the elements within each orbit of the so called *diagonal* $\mathcal{S}_N$-action on $\Lambda$; see Fig. 4.1 (left, middle).

## 4.1 Pool & Broadcast Interpretation

Each unique parameter in the constraint $\Lambda$ corresponds to a linear operation that has a pool and broadcast interpretation. Moreover this interpretation allows for a linear-time implementation of the equivariant layers, as we avoid the explicit construction of $\Lambda$.

**Definition 2** (Pooling)**.** *For $\mathbb{P} = \{p_1, \ldots, p_L\} \subseteq [M]$, the pooling operation sums over the indices in $\mathbb{P}$:*

$$\mathrm{Pool}_{\mathbb{P}}(\mathbf{X}_{\delta_1\ldots\delta_M}) = \sum_{\delta_{p_1}\in[N]} \cdots \sum_{\delta_{p_L}\in[N]} \mathbf{X}_{\delta_1,\ldots,\delta_M} \tag{4.4}$$

Note that in practice the summation in the definition may be replaced with any permutation-invariant aggregation function. We use mean-pooling in our experiments.

**Definition 3** (Broadcasting)**.** $\mathrm{Bcast}_{\mathbb{B},M'}(\mathbf{X})$ *broadcasts a* source *tensor* $\mathbf{X}$ *of order $M$ over a target tensor of order $M' \geq M$. We identify* $\mathbf{X}$ *with a sequence of dimensions of the target tensor $\mathbb{B} = (b_1, \ldots, b_M)$ with $b_m \in [M']$, and we broadcast across the remaining $M' - M$ dimensions – that is*

$$\left(\mathrm{Bcast}_{\mathbb{B},M'}(\mathbf{X})\right)_{\delta_1,\ldots,\delta_{M'}} = \mathbf{X}_{\delta_{b_1},\ldots,\delta_{b_M}} \tag{4.5}$$

For example, with $\mathbf{X} \in R^{N\times N}$, $\mathrm{Bcast}_{\langle 0,1\rangle,3}(\mathbf{X})$ maps $\mathbf{X}$ across the first two axis of a three-dimensional tensor and broadcasts its values along the third dimension.

The important fact about pool and broadcast operations defined above is that they are equivariant to permutation of nodes. Indeed, we can write the linear operations within $\Lambda$ as different combinations of pooling and broadcasting of input incidence tensor into an output tensor. More specifically, first pool the input tensor in all possible ways, and then broadcast the resulting collection of pooled tensors to the target tensor, again in all ways possible. Each unique combination of pooled object and broadcasting target receives its own unique parameter $\lambda_{\mathbb{B},\mathbb{P}}$ – that is

$$\Lambda^{M\to M'}(\mathbf{X}) = \sum_{\substack{\mathbb{P}\subseteq[M]\\ \mathbb{B}\subseteq\langle 1,\dots,M'\rangle\\ |\mathbb{B}|=M-|\mathbb{P}|}} \lambda_{\mathbb{B},\mathbb{P}}\,\mathrm{Bcast}_{\mathbb{B},M'}\left(\mathrm{Pool}_{\mathbb{P}}(\mathbf{X})\right) \tag{4.6}$$

**Claim 1.** *The number of independent operations $\tau^{M\to M'}$ in Eq.* (4.6) *it is given by*

$$\tau^{M\to M'} = \sum_{m=0}^{\min(M,M')} \binom{M}{m}\binom{M'}{m} m!. \tag{4.7}$$

*Proof.* We can extract tensors of order $m \le \min(M,M')$ by pooling over $M-m$ of the input indices. There are $\binom{M}{M-m} = \binom{M}{m}$ different ways of pooling for each value of $m$. Then, we rebroadcast the pooled tensor of order $m$ to the target tensor of order $M'$. There are $M'!/(M'-m)!$ unique ways of broadcasting. □

### 4.1.1 Additional Symmetry of Undirected Faces

When counting the number of unique pooling and broadcasting operations, so far we assumed that $\mathbf{X}_{\delta^{(M)}} = \mathbf{X}_{(\delta_1,\dots,\delta_M)} \neq \mathbf{X}_{(\pi\cdot\delta_1,\dots,\pi\cdot\delta_M)}$. However, if $\delta^{(M)}$ is an undirected face, the face-vector is invariant to permutation of its nodes. This symmetry reduces the number of independent parameters in Eq. (4.6). Furthermore, if we enforce the same symmetry on the output tensor, some of the weights need to be tied together. In particular, there is only one tensor of order $m$ that can be extracted through pooling for each value of $0 \le m \le \min(M,M')$. Similarly, all possible ways of broadcasting the pooled tensor to the target tensor have to carry the same parameter in order to restore symmetry of the output. Thus, in comparison to Eq. (4.7), for the symmetric input and symmetric output case, the degrees of freedom of the equivariant map are significantly reduced:

$$\tau^{(M\to M')}_{\mathrm{symm}} = \sum_{m=0}^{\min(M,M')} 1 = \min(M,M') + 1. \tag{4.8}$$

## 4.2 Decomposition of Equivariant Maps

Let $\Lambda : (\biguplus_m \kappa_m \mathbf{X}_{\delta^m}) \mapsto (\biguplus_{m'} \kappa_{m'} \mathbf{X}_{\delta^{m'}})$ be an equivariant map between arbitrary incidence tensors, where both input and output decompose according to Eq. (3.2). Using the equivariant maps $\Lambda^{m\to m'}$ of Eq. (4.6), we get a decomposition of $\Lambda$.[1]

[1] Note that this decomposition remains valid when using $\mathcal{A}ut(\mathbf{X}) < \mathcal{S}_N$, as the orbits of $\mathcal{S}_N$ remain invariant for all its subgroups.

$$\Lambda\big(\mathbf{X}_{\delta_1,\dots,\delta_{D'}|\Sigma}\big) \cong \bigcup_{m'} \bigcup_{k'=1}^{\kappa'_m} \sum_{m} \sum_{k=1}^{\kappa_m} \Lambda^{k,m\to k',m'}\big(\mathbf{X}_{\delta^{(m)}}\big), \tag{4.9}$$

where for each copy (out of $\kappa'_m$ copies) of the output face of size $m$, we are summing over all the maps produced by different input faces having different multiplicities. Use of $k$ and $k'$ in the map $\Lambda^{k,m\to k',m'}$ is to indicate that for each input-output copy, the map $\Lambda^{m\to m'}$ uses a different set of parameters. The upshot is that input and output multiplicities play a role similar to input and output *channels*. From Eq. (4.9) it follows that the total number of independent parameters in a layer is

$$\tau = \sum_{m,m'} \kappa_{m'}\kappa_m \tau^{m\to m'}, \tag{4.10}$$

where $\tau^{m\to m'}$ is given by Eq. (4.7).

**Example 3** (Node-adjacency tensors)**.** *This example, is concerned with the incidence representation used in equivariant graph networks of Maron et al. [27] and derives their model as a special case, using our pool/broadcast layer and face-vectors decomposition. For equivariant layer that maps a node-node-…-node incidence tensor* $\mathbf{X}$ *of order* $M$ *(as outlined in Example 2) to the same structure, the decomposition in terms of face-vectors reads*

$$\mathbf{X}_{\delta_1,\dots,\delta_M} \cong \bigcup_{m} \left\{{M \atop m}\right\} \mathbf{X}_{\delta^{(m)}}, \tag{4.11}$$

*where* $\left\{{M \atop m}\right\}$ *is the Stirling number of the second kind; see Example 2. The total number of operations according to Eq.* (4.10) *is then given by*

$$\tau = \sum_{m,m'=1}^{M} \left\{{M \atop m}\right\}\left\{{M \atop m'}\right\} \sum_{l=0}^{\min(m,m')} \binom{m}{l}\binom{m'}{l} l! \tag{4.12}$$

$$= \sum_{l=0}^{M}\sum_{m=l}^{M}\sum_{m'=l}^{M} \left[\binom{m}{l}\left\{{M \atop m}\right\}\right]\left[\binom{m'}{l}\left\{{M \atop m'}\right\}\right] l! = \mathrm{Bell}(2M). \tag{4.13}$$

*In the last line,* $\mathrm{Bell}(2M)$ *is the Bell number and counts the number of unique partitions of a set of size* $2M$*. To see the logic in the final equality: first divide* $[2M]$ *in half. Next, partition each half into partitions of different sizes (*$0 \le m, m' \le M$*) and choose* $l$ *of these partitions from each half and merge them in pairs. The first two terms count the number of ways we can partition each half into* $m$ *(or* $m'$*) partitions and select a subset of size* $l$ *among them. The* $l!$ *term accounts for different ways in which* $l$ *partitions can be aligned. This result agrees with the result of [27].*

**Example 4** (Symmetric node-adjacency tensors)**.** *Consider a similar setup, but for the case of undirected faces. In this case we map a symmetric input into a symmetric output. From Eq.* (4.10)*, where we use*

*now Eq.* (4.8) *for the symmetric case, we get*

$$\tau = \sum_{m,m'=1}^{M} (\min(m,m')+1) = \frac{1}{6}(2M^3+9M^2+M). \tag{4.14}$$

*Note that we have omitted the multiplicity coefficients $\kappa_{m,m'}$, assuming that all faces of a given dimension are equal.*

### 4.2.1 Representation and Expressiveness

Consider a simplicial complex or polytope where faces of particular dimension have associated attributes. This information may be directly represented using face-vectors $\biguplus_{m=1}^{M} \mathbf{X}_{\delta^{(m)}}$. Alternatively, we may only use the largest face $\delta^{(M)}$, and broadcast all lower dimensional data to the maximal face-vectors. This gives an equivalent representation

$$\bigcupdot_{m=1}^{M} \mathbf{X}_{\delta^{(m)}} \equiv \bigcupdot_{m=1}^{M} \mathrm{Bcast}_{\mathbb{B},M} \mathbf{X}_{\delta^{(m)}} = \bigcupdot_{k=1}^{M} \mathbf{X}^{k}_{\delta^{(M)}}, \tag{4.15}$$

that resembles having multiple channels, indexed by $k$.

Yet another alternative is to use an incidence tensor $\mathbf{X}_{\delta_1,\ldots,\delta_M|\Sigma} \cong \biguplus_{m=1}^{M} \kappa_m \mathbf{X}_{\delta^{(m)}}$ that decomposes into face vectors according to Theorem 3.2.1. We have similarly diverse alternatives for the "output" of an equivariant map. *Observe that the corresponding equivariant maps have the same expressiveness up to change in the number of channels.* This is because we could produce the same pool-broadcast features of Eq. (4.6) across different representations.

**Example 5** (Node-node vs node-edge incidence)**.** *Recall that the node-node incidence for an undirected graph decomposes as $\mathbf{X}_{\{\delta_1\},\{\delta_2\}} \cong \mathbf{X}_{\{\delta_1\}} \uplus \mathbf{X}_{\{\delta_1,\delta_2\}}$, where $\mathbf{X}_{\{\delta_1,\delta_2\}}$ is in turn isomorphic to the node-edge incidence matrix $\mathbf{X}_{\{\delta_1\},\{\delta_1,\delta_2\}}$. We can also broadcast a node-vector to a node-edge matrix $\tilde{\mathbf{X}}_{\{\delta_1\},\{\delta_1,\delta_2\}} \leftarrow \mathrm{Bcast}_{\langle 1\rangle,2}(\mathbf{X}_{\{\delta_1\}})$ to get a node-edge incidence tensor, with two channels $\mathbf{X}_{\{\delta_1\},\{\delta_1,\delta_2\}}$ and $\tilde{\mathbf{X}}_{\{\delta_1\},\{\delta_1,\delta_2\}}$, that represents the same set of node and edge attributes as the node-node incidence $\mathbf{X}_{\{\delta_1\},\{\delta_2\}}$. The corresponding equivariant layers visualized in Fig. 4.1 (left, middle) have the same expressiveness. Note that we are doubling the number of parameters (due to having two input channels) for the node-edge layer.*

## 4.3 Sparse Tensors and Non-Linear Layers

The equivariant layers in Section 4.2 require a completion of the incidence tensor (e.g., a fully connected graph). To avoid the cost of a dense representation, one may apply a sparsity mask after the linear map, while preserving equivariance:

$$\Lambda_{\mathrm{sp}} : \mathbf{X} \to \Lambda(\mathbf{X}) \circ s(\mathbf{X}), \tag{4.16}$$

where $\Lambda$ is the equivariant linear map of Section 4.2, $s(\mathbf{X})$ is the sparsity mask, and $\circ$ is the Hadamard product. For example, if the layer output has the same shape as the input, one might choose to preserve

the sparsity of the input. In this case, $s(\mathbf{X})$ will have zero entries where the input $\mathbf{X}$ has zero entries, and ones otherwise. However, the setting of Eq. (4.16) is more general as input and output may have different forms. Since the sparsity mask $s(\mathbf{X})$ depends on the input, the map of Eq. (4.16) is now non-linear. In practice, rather than calculating the dense output and applying the sparsity mask, we directly produce the non-zero values.

## 4.4 Further Relaxation of The Symmetry Group

The neural layers discussed so far are equivariant to the group $\mathcal{G} = \mathcal{S}_N$ where $N$ is the number of nodes. The action of $\mathcal{S}_N$ on on higher-dimensional faces is naturally defined in Eq. (3.1). A simplifying alternative is to assume an independent permutation for each dimension of the incidence tensor. Let $N_d = |\{\boldsymbol{\delta}^{(d)}\}|$ denote the number of faces of size $d$, so that $N_1 = N$. Consider the action of $\boldsymbol{\pi} = (\boldsymbol{\pi}^1 \dots, \boldsymbol{\pi}^M) \in \mathcal{S}_{N_{|\delta_1|}} \times \dots, \mathcal{S}_{N_{|\delta_M|}}$ on $\mathbf{X}_{\delta_1,\dots,\delta_M}$:

$$\boldsymbol{\pi} \cdot \mathbf{X}_{\delta_1,\dots,\delta_M} = \mathbf{X}_{\pi^1 \cdot \delta_1,\dots,\pi^M \cdot \delta_M}, \tag{4.17}$$

where $\boldsymbol{\pi}^m \cdot \boldsymbol{\delta}_m$ is one of $N_{|\delta_m|}!$ permutations of these faces. The corresponding equivariant layer introduced in [14] has $\tau = 2^M$ unique parameters, and it is relatively easy to implement.

In Appendix A we show how to construct such $(\mathcal{S}_N \times \mathcal{S}_{N_2})$-equivariant sparsity-preserving (and therefore non-linear) layer for a node-edge incidence matrix. Even though a single layer is over-constrained by these symmetry assumptions, we prove that two such layers are enough to generate the same node and edge features of a single linear layer for a node-node incidence. These results are corroborated by good performance on experiments (see **HSD̄E** in Table 5.2).

# Chapter 5

# Experiments

In this chapter, we provide our evaluation results on QM9 dataset[31]. The details of the dataset can be found in Section 5.1. We used Pytorch to implement and train our models. In Section 5.2 we discuss the model architecture, training procedure, and other implementation details. Finally, Section 5.3 reports experimental results along with a performance comparison with other related models.

## 5.1 Dataset

QM9 dataset contains 133,885 small organic molecules consist of Hydrogen (H), Carbon (C), Oxygen (O), Nitrogen (N), and Flourine (F) atoms and contain up to 9 heavy (non Hydrogen) atoms and up to 29 atoms in total including Hydrogen. Each molecule is represented as graph with a symmetric adjacency matrix. There are a number of features available for each atom in a molecule including atomic coordinates, atom type, atomic numbers, etc. We use all these atomic features as node features in the input graph representation. Moreover, We use edge length (the distance between each two atoms) along with one-hot representation of bond type (single, double, triple, or aromatic) as edge features in the graph. For each molecule, 12 target chemical properties are calculated at the B3LYP/6-31G(2df,p) level of quantum chemistry. The targets are listed in table 5.1. We perform regression task on these targets and evaluate mean absolute error.

## 5.2 Architecture & Training Procedure

Our architecture for all models is a simple stack of equivariant layers:

$$\mathrm{Pool}_{\{0,1\}}\Lambda^{(\ell)}(\mathrm{ReLU}(\Lambda^{(\ell-1)}\ldots\mathrm{ReLU}(\Lambda^{(1)}\mathbf{X}))\ldots),$$

where the final layer has a single channel followed by pooling, which produces a scalar value for the target.

Following [11], we randomly chose 10000 samples for validation, 10000 samples for testing, and used the rest for training. All targets were normalized to have mean 0 and variance 1. We perform regression task on these targets, with the same training-test split as competition, reporting the mean

**Table 5.1:** Target Molecular Properties

| TARGET | UNIT | DESCRIPTION |
|---|---|---|
| $\alpha$ | $Bohr^3$ | Isotropic polarizability |
| $C_v$ | $cal/molK$ | Heat capacity at 298.15K |
| G | $eV$ | Free energy at 298.15K |
| H | $eV$ | Enthalpy at 298.15K |
| $\varepsilon_{HOMO}$ | $eV$ | Energy of highest occupied molecular orbital |
| $\varepsilon_{LUMO}$ | $eV$ | Energy of lowest occupied molecular orbital |
| $\Delta_\varepsilon$ gap | $eV$ | Difference between $\varepsilon_{HOMO}$ and $\varepsilon_{LUMO}$ |
| $\mu$ | $Debye$ | dipole moment |
| $\langle R_2 \rangle$ | $Bohr^2$ | Electronic spatial extent |
| U | $eV$ | Internal energy at 298.15K |
| $U_0$ | $eV$ | Internal energy at 0K |
| ZPVE | $eV$ | Zero point vibrational energy |

absolute error (MAE) on each target.

We trained one model per target and performed (non-exhaustive) hyper parameter search for the number of layers $3 \le \ell \le 20$, channels in $\{128, 256\}$, and batch size in $\{16, 32, 64, 128\}$. We used ADAM [20] with an initial learning rate of $10^{-3}$ and adaptive learning rate decay based on validation error. We trained each model for 1000 epochs and minimized the mean squared error between the model output and the target. A small weight decay of $10^{-6}$ was used. During training, for every molecule in the mini-batch we randomly rotate atom coordinates with uniform distribution along z axis in each epoch. We found that batch-normalization [16] to be very effective in accelerating the training of incidence networks. The number of layers and channels-per-layer, as well as the mini-batch size are treated as hyper-parameters.

## 5.3 Experimental Results

As a surrogate for *density function theory*, many deep models for graphs have been applied to the task of predicting molecular properties [1, 11, 17, 23, 30, 34, 35, 38]; interestingly, most, if not all of these methods are considered message passing methods.[1] A drawback of a fully-fledged message passing scheme compared to incidence networks is its scalability. However, this is not an issue for QM9 dataset [31] that contains 133,885 "small" organic molecules.

Table 5.2 reports previous state-of-the-art, as well as our results using various members of the incidence network family. The abbreviation used for the results include: Sparse vs. Dense (**S/$\bar{\text{S}}$**); directed vs. undirected (**D/$\bar{\text{D}}$**) and node-edge vs. node-node (**E/$\bar{\text{E}}$**). For example, **SD$\bar{\text{E}}$** uses sparse nonlinear layers that operate on directed node-node adjacency and produce directed asymmetric outputs. Finally (**H**) identifies the layer that uses the larger $\mathcal{S}_N \times \mathcal{S}_{N_2}$ symmetry. See Table 5.3 for more details on each

[1] We were not able to compare our experimental results to [28, 30] and the results reported in [39] due to their choice of using a larger training split. Moreover, the raw QM9 dataset used by [30] contains 133,246 molecules, which has 639 fewer molecules than the dataset used in our experiments.

incidence network model.

**Table 5.2:** Mean absolute errors on the QM9 targets. ENN-S2S is the neural message passing of Gilmer et al. [11] and NMP-EDGE [17] is its improved variation with edge updates. SCHNET uses a continuous filter convolution operation Schütt et al. [35]. CORMORANT uses a rotation equivariant architecture [1]. The results of WAVESCATT [15] were taken from [1]. Results where an incidence network achieves state-of-the-art is in bold.

| | | | | | | INCIDENCE NETWORKS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TARGET | ENN-S2S | NMP-EDGE | SCHNET | CORMORANT | WAVESCATT | $\mathbf{S\bar{D}\bar{E}}$ | $\mathbf{SD\bar{E}}$ | $\mathbf{S\bar{D}\bar{E}}$ | $\mathbf{\bar{S}D\bar{E}}$ | $\mathbf{SD\bar{E}}$ | $\mathbf{HS\bar{D}E}$ |
| $\alpha$ | 0.092 | 0.077 | 0.235 | 0.092 | 0.160 | **0.039** | **0.030** | **0.036** | **0.037** | **0.033** | **0.033** |
| $C_v$ | 0.040 | 0.032 | 0.033 | 0.031 | 0.049 | **0.025** | **0.028** | **0.030** | **0.023** | **0.028** | **0.029** |
| G | 0.019 | 0.012 | 0.014 | - | - | **0.001** | **0.008** | **0.008** | **0.003** | **0.011** | **0.010** |
| H | 0.017 | 0.011 | 0.014 | - | - | **0.001** | **0.008** | **0.008** | **0.002** | **0.010** | **0.010** |
| $\varepsilon_{HOMO}$ | 0.043 | 0.036 | 0.041 | 0.036 | 0.085 | 0.191 | 0.089 | 0.116 | 0.097 | 0.101 | 0.090 |
| $\varepsilon_{LUMO}$ | 0.037 | 0.030 | 0.034 | 0.036 | 0.076 | 0.062 | 0.049 | 0.052 | 0.054 | 0.054 | 0.052 |
| $\Delta_\varepsilon$ gap | 0.069 | 0.058 | 0.063 | 0.073 | 0.118 | 0.062 | 0.068 | 0.080 | 0.087 | 0.078 | 0.071 |
| $\mu$ | 0.030 | 0.029 | 0.033 | 0.130 | 0.340 | 0.082 | 0.040 | 0.067 | 0.038 | 0.055 | 0.060 |
| $\langle R_2 \rangle$ | 0.180 | 0.072 | 0.073 | 0.673 | 0.410 | **0.012** | **0.017** | **0.017** | **0.009** | **0.021** | **0.017** |
| U | 0.019 | 0.010 | 0.019 | - | - | **0.002** | **0.007** | **0.009** | **0.002** | **0.010** | **0.009** |
| $U_0$ | 0.019 | 0.010 | 0.014 | 0.028 | 0.022 | **0.001** | **0.008** | **0.008** | **0.003** | **0.010** | **0.010** |
| ZPVE | 0.0015 | 0.0014 | 0.0017 | 0.0019 | 0.002 | 0.008 | 0.008 | 0.011 | 0.007 | 0.010 | 0.009 |

All models match or outperform state-of-the-art in 7/12 targets (bold values). They also show a similar performance despite using different representations, supporting our theoretical analysis regarding the comparable expressiveness node-node and node-edge representation. Dense models generally perform slightly better at the cost of 3× run-time for training. Finally, we note that the 4 parameter model (**HSD̄E**) of Section 4.4 performs almost as well, despite using an over-constraining symmetry group, further supporting our theoretical results outlined in Section 4.4 and explained in Appendix A.

**Table 5.3:** Details of the layers used in the experiments, and Table 5.2. All layers except the last one are equivariant to the symmetry group $\mathcal{S}_N$ where $N$ is the number nodes and the last layer is equivariant to the group $\mathcal{S}_N \times \mathcal{S}_{N_2}$ where $N_2$ is the number of faces (edges).

| | Layer Name | Edge Type | Graph Type | Num. Params. | Layer Type | Sym. Group |
|---|---|---|---|---|---|---|
| **Node-Node** | $\mathbf{\bar{S}\bar{D}\bar{E}}$ | Undirected | Dense | 9 | Linear | $\mathcal{S}_N$ |
| | $\mathbf{SD\bar{E}}$ | Directed | Sparse | 15 | Non-Linear | $\mathcal{S}_N$ |
| | $\mathbf{S\bar{D}\bar{E}}$ | Undirected | Sparse | 9 | Non-Linear | $\mathcal{S}_N$ |
| **Node-Edge** | $\mathbf{\bar{S}\bar{D}E}$ | Undirected | Dense | 7 | Linear | $\mathcal{S}_N$ |
| | $\mathbf{S\bar{D}E}$ | Undirected | Sparse | 7 | Non-Linear | $\mathcal{S}_N$ |
| | $\mathbf{HS\bar{D}E}$ | Undirected | Sparse | 4 | Non-Linear | $\mathcal{S}_N \times \mathcal{S}_{N_2}$ |

# Chapter 6

# Conclusion and future work

This thesis introduces a general approach to learning equivariant models for a large family of structured data through their incidence tensor representation. In particular, we showed various incidence tensor representations for graphs, simplicial complexes, and abstract polytopes. The proposed family of incidence networks are 1) modular: they decompose to simple building blocks; 2) efficient: they all have linear-time pooling-broadcasting implementation, and; 3) effective: various members of this family achieve state-of-the-art performance using simple a architecture.

In our systematic study of this family, we discussed implications of 1) added symmetry due to undirected faces; 2) sparsity preserving equivariant maps, and; 3) the successive relaxation of the symmetry group $\mathcal{A}ut(\mathbf{X}) \leq \mathcal{S}_N \leq \mathcal{S}_{N_1} \times \ldots \times \mathcal{S}_{N_M}$. Here, moving to a larger group simplifies the neural layer by reducing the number of unique parameters (and linear operations), while increasing its bias. Application of incidence networks to different domains, such as learning on triangulated mesh, is a direction that we hope to explore in the future.

# Bibliography


[1] B. Anderson, T.-S. Hy, and R. Kondor. Cormorant: Covariant molecular neural networks. *arXiv preprint arXiv:1906.04015*, 2019. → pages viii, 19, 20

[2] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. → page 3

[3] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. → page 4

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *ICLR*, 2014. → page 4

[5] Z. Chen, S. Villar, L. Chen, and J. Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *arXiv preprint arXiv:1905.12560*, 2019. → page 6

[6] T. S. Cohen and M. Welling. Group equivariant convolutional networks. *arXiv preprint arXiv:1602.07576*, 2016. → page 5

[7] T. S. Cohen, M. Geiger, and M. Weiler. Intertwiners between induced representations (with applications to the theory of equivariant neural networks). *arXiv preprint arXiv:1803.10743*, 2018. → page 5

[8] T. S. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling. Gauge equivariant convolutional networks and the icosahedral cnn. *arXiv preprint arXiv:1902.04615*, 2019. → page 6

[9] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016. → page 4

[10] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, 2015. → page 4

[11] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017. → pages viii, 4, 18, 19, 20

[12] D. Graham and S. Ravanbakhsh. Deep models for relational databases. *arXiv preprint arXiv:1903.09033*, 2019. → page 6

[13] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017. → page 3

[14] J. Hartford, D. R. Graham, K. Leyton-Brown, and S. Ravanbakhsh. Deep models of interactions across sets. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1909–1918, 2018. → pages 5, 17, 26

[15] M. Hirn, S. Mallat, and N. Poilvert. Wavelet scattering regression of quantum chemical energies. *Multiscale Modeling & Simulation*, 15(2):827–863, 2017. → pages viii, 20

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. → page 19

[17] P. B. Jørgensen, K. W. Jacobsen, and M. N. Schmidt. Neural message passing with edge updates for predicting properties of molecules and materials. *arXiv preprint arXiv:1806.03146*, 2018. → pages viii, 19, 20

[18] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016. → page 4

[19] N. Keriven and G. Peyré. Universal invariant and equivariant graph neural networks. *arXiv preprint arXiv:1905.04943*, 2019. → page 6

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. → page 19

[21] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. → page 4

[22] R. Kondor and S. Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. *arXiv preprint arXiv:1802.03690*, 2018. → page 5

[23] R. Kondor, H. T. Son, H. Pan, B. Anderson, and S. Trivedi. Covariant compositional networks for learning graphs. *arXiv preprint arXiv:1801.02144*, 2018. → page 19

[24] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. → page 5

[25] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1): 97–109, 2018. → page 4

[26] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015. → page 4

[27] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018. → pages 5, 15

[28] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019. → page 19

[29] H. Maron, E. Fetaya, N. Segol, and Y. Lipman. On the universality of invariant networks. *arXiv preprint arXiv:1901.09342*, 2019. → page 6

[30] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *arXiv preprint arXiv:1810.02244*, 2018. → page 19

[31] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1:140022, 2014. → pages 18, 19

[32] S. Ravanbakhsh, J. Schneider, and B. Poczos. Equivariance through parameter-sharing. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *JMLR: WCP*, August 2017. → page 5

[33] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. → page 3

[34] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8:13890, 2017. → pages 4, 19

[35] K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller. Schnet–a deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24): 241722, 2018. → pages viii, 19, 20

[36] J. Shawe-Taylor. Building symmetries into feedforward networks. In *1989 First IEE International Conference on Artificial Neural Networks,(Conf. Publ. No. 313)*, pages 158–162. IET, 1989. → page 5

[37] J. Shawe-Taylor. Symmetries and discriminability in feedforward network architectures. *IEEE Transactions on Neural Networks*, 4(5):816–826, 1993. → page 5

[38] O. T. Unke and M. Meuwly. Physnet: A neural network for predicting energies, forces, dipole moments and partial charges. *arXiv preprint arXiv:1902.08408*, 2019. → page 19

[39] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2): 513–530, 2018. → page 19

[40] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, 2017. → pages 5, 10

# Appendix A

# Supporting Materials

## A.1 Additional Results for Node-Edge Layers

Consider an undirected graph with $N$ nodes and its node-node $\mathbf{X}_{\{\delta_1\},\{\delta_2\}}$ and node-edge $\mathbf{X}_{\{\delta_1\},\{\delta_1,\delta_2\}}$ incidence representations. We discussed the equivalence of their $\mathcal{S}_N$-equivariant linear layers in Example 5. Here, we study node-edge layers equivariant to $\mathcal{S}_N \times \mathcal{S}_{N_2}$, where $N_2 = N(N-1)/2$ is the number of edges, and compare their expressive power with their $\mathcal{S}_N$-equivariant linear counterparts.

NODE-NODE INCIDENCE. Let $\mathbf{A} \in R^{N\times N\times K}$ be a *node-node* incidence matrix with $K$ channels. Consider a linear layer $\Lambda_A : \mathbf{A} \mapsto \mathbf{A}' \in R^{N\times N\times K'}$, where $\Lambda_A$ is defined as in Eq. (4.9), and $K'$ are output channels. Diagonal elements of $\mathbf{A}$ encode input node features and off-diagonal elements input edge features. Since the graph is undirected, $\mathbf{A}$ is symmetric, and the corresponding number of independent pooling and broadcasting operations from Eq. (4.14) is $\tau_A = 9$, for a total of $9KK'$ parameters.

NODE-EDGE INCIDENCE. Alternatively, one could represent the graph with a *node-edge* incidence matrix $\mathbf{B} \in R^{N\times N_2\times 2K}$. Node features are mapped on the first $K$ channels along the node dimension and broadcasted across the edge dimension. Similarly, edge features are mapped on the last $K$ channels along the edge dimension and broadcasted across the node dimension. Consider a layer equivariant to $\mathcal{S}_N \times \mathcal{S}_{N_2}$ as described in Section 4.4, that also preserves the sparsity through the non-linear implementation of Section 4.3, $\Lambda_B : \mathbf{B} \mapsto \mathbf{B}' \in R^{N\times N_2\times 2K'}$, where $\mathbf{B}' = \bar{\Lambda}(\mathbf{B}) \circ s(\mathbf{B})$, and $\bar{\Lambda}$ is an equivariant map defined as in [14]. We can write this linear map in our notation as

$$\bar{\Lambda}(\mathbf{B}) = \sum_{\mathbb{P}\in 2^{\{0,1\}}} \lambda_{\mathbb{P}}\, \mathrm{Bcast}_{\{0,1\}-\mathbb{P}}\, \mathrm{Pool}_{\mathbb{P}}\, \mathbf{B}, \tag{A.1}$$

where $2^{\{0,1\}}$ is the set of all subsets of $\{0,1\}$, and we have dropped the output dimension in the Bcast operator, as all features are broadcasted back to $\mathbf{B}'$. $\bar{\Lambda}(\mathbf{B})$ corresponds to pooling/broadcasting each of the two dimensions of $\mathbf{B}$ independently, thus the summation has four terms, for pooling/broadcasting over rows, columns, both rows and columns and no pooling at all. The number of independent operations

is $\tau_B = 4$ for a total of $16KK'$ independent parameters.

**Theorem A.1.1.** *Let $\Lambda_A$ and $\Lambda_B$ be the $\mathcal{S}_N$-equivariant and $(\mathcal{S}_N \times \mathcal{S}_{N_2})$-equivariant layers operating on node-node and node-edge incidence, respectively. The following statements hold:*

*(a) a single $\Lambda_B$ layer spans a subspace of features spanned by $\Lambda_A$,*

*(b) two $\Lambda_B$ layers span the same feature space spanned by $\Lambda_A$ (maximal feature space).*

*Proof.* First, we discuss how to interpret output features. Additionally, for the rest of the proof we will assume $K = K' = 1$ for simplicity, noting that the proof generalizes to the multi-channel case.

*Output Features.* For a node-node incidence layer, it is natural to interpret diagonal and off-diagonal elements of $\mathbf{A}'$ as output node and edge features, respectively.

For the node-edge incidence case, all four operations of the $\Lambda_B$ map return linear combinations of features that vary at most across one dimension, and are repeated across the remaining dimensions. This is the same pattern of input node and edge features, and we will use the same scheme to interpret them. In particular,

- an output feature that varies across the node dimension (but it is repeated across the edge dimension) is a node feature,
- an output feature that varies across the edge dimension (but it is repeated across the node dimension) is an edge feature,
- and finally a feature that is repeated across both node and edge dimension is either a node or edge feature.

For example, consider a complete graph with three nodes. Its incidence matrix with node and edge features $\eta_{\delta_1}$ and $\varepsilon_{\delta_1,\delta_2}$ repeat across rows and columns as follows

$$\mathbf{B} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} \{12\} & \{13\} & \{23\} \end{array} \\ \begin{pmatrix} (\textcircled{\eta_1}, \boxed{\varepsilon_{12}}) & (\eta_1, \boxed{\varepsilon_{13}}) & - \\ (\textcircled{\eta_2}, \varepsilon_{12}) & - & (\eta_2, \boxed{\varepsilon_{23}}) \\ - & (\textcircled{\eta_3}, \varepsilon_{13}) & (\eta_3, \varepsilon_{23}) \end{pmatrix} \end{array}, \tag{A.2}$$

$\square$ = edge features,

$\bigcirc$ = node features.

Consider pooling and broadcasting across the edge dimension. Node features were broadcasted across it, and since every node is incident with two edges, we get back multiples of the original features. The edge channel will instead return new features that combine the edges incident on each node. These new node features vary across the node dimension and are broadcasted across the edge dimension,

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} \{12\} & \{13\} & \{23\} \end{array} \\ \begin{pmatrix} (2\eta_1, \overbrace{\varepsilon_{12}+\varepsilon_{13}}) & (2\eta_1, \varepsilon_{12}+\varepsilon_{13}) & - \\ (2\eta_2, \overbrace{\varepsilon_{12}+\varepsilon_{23}}) & - & (2\eta_2, \varepsilon_{12}+\varepsilon_{23}) \\ - & (2\eta_3, \overbrace{\varepsilon_{13}+\varepsilon_{23}}) & (2\eta_3, \varepsilon_{13}+\varepsilon_{23}) \end{pmatrix} \end{array}. \tag{A.3}$$

On the other hand, pooling and broadcasting across the node dimension returns multiples of input edge features, and generates new edge features from the two nodes incident on each edge,

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} \{12\} & \{13\} & \{23\} \end{array} \\ \begin{pmatrix} (\boxed{\eta_1+n_2}, 2\varepsilon_{12}) & (\boxed{\eta_1+\eta_3}, 2\varepsilon_{13}) & - \\ (\eta_1+\eta_2, 2\varepsilon_{12}) & - & (\boxed{\eta_2+\eta_3}, 2\varepsilon_{23)} \\ - & (\eta_1+\eta_3, 2\varepsilon_{13}) & (\eta_2+\eta_3, 2\varepsilon_{23}) \end{pmatrix} \end{array}. \tag{A.4}$$

*Proof of statement (a).* Given the feature encoding described above, $\Lambda_A$ and $\Lambda_B$ layers can be represented as a function acting on the space of node and edge features,

$$\mathbf{L} : R^{N_G} \mapsto R^{N_G}, \tag{A.5}$$

where $N_G = N(N+1)/2$ is the number of graph elements, i.e., the sum of nodes and edges. Let us fix a basis in $R^{N_G}$ such that the first $N$ components of a vector $\phi \in R^{N_G}$ represent node features and the remaining $N_2 = N(N-1)/2$ represent edge features

$$\phi = \big( \underbrace{\eta_1, \ldots, \eta_N}_{\text{node features}}, \underbrace{\varepsilon_1, \ldots, \varepsilon_{N_e}}_{\text{edge features}} \big)^T \equiv \big(\eta, \varepsilon\big)^T. \tag{A.6}$$

Then a layer has a matrix representation $\mathbf{L} \in R^{N_G \times N_G}$,

$$\phi \mapsto \mathbf{L}\phi = \begin{pmatrix} \mathbf{L}_{\langle nn \rangle} & \mathbf{L}_{\langle ne \rangle} \\ \mathbf{L}_{\langle en \rangle} & \mathbf{L}_{\langle ee \rangle} \end{pmatrix} \begin{pmatrix} \eta \\ \varepsilon \end{pmatrix}, \tag{A.7}$$

where we have split the matrix into four sub-blocks acting on the sub-vectors of node and edge features. In particular, they represents the following classes of operators: $\mathbf{L}_{\langle nn \rangle} \in R^{N \times N}$ maps input node features to output node features, $\mathbf{L}_{\langle ne \rangle} \in R^{N \times N_2}$ maps input edge features to output node features, $\mathbf{L}_{\langle en \rangle} \in R^{N_2 \times N}$ maps input node features to output edge features, and $\mathbf{L}_{\langle ee \rangle} \in R^{N_2 \times N_2}$ maps input edge features to output edge features.

The nine operations of the $\Lambda_A$ map can be written as

$$\Lambda_A \simeq \begin{pmatrix} \mathbf{L}_{\langle nn\rangle,0} + \mathbf{L}_{\langle nn\rangle,1} & \mathbf{L}_{\langle ne\rangle,0} + \mathbf{L}_{\langle ne\rangle,1} \\ \mathbf{L}_{\langle en\rangle,0} + \mathbf{L}_{\langle en\rangle,1} & \mathbf{L}_{\langle ee\rangle,0} + \mathbf{L}_{\langle ee\rangle,1} + \mathbf{L}_{\langle ee\rangle,2} \end{pmatrix}, \tag{A.8}$$

and are summarized in Table A.1. We have split the operations according to the sub-blocks defined above, with $\mathbf{L}_{\langle ab\rangle,i}$ labeling the operator that maps input $b$ features into output $a$ features (with $a$ and $b$ labeling either node or edges), and with $i$ representing the dimension of the pooled tensor. For example, $\mathbf{L}_{\langle nn\rangle,0}$ is the operator that pools all node features (i.e., it pools the vector of node features to dimension zero) and broadcasts the pooled tensor over nodes. The symbol $\simeq$ indicates that each operator is defined up to a multiplicative constant (i.e., the corresponding learnable parameter $\lambda$ in Eq. (4.6)). The action of $\Lambda_A$ on the space of node and edge features can be uniquely identified by the nine-dimensional subspace

$$V_{\Lambda_A} = \text{span}\left(\left\{\begin{pmatrix} \mathbf{L}_{\langle nn\rangle,0} & 0 \\ 0 & 0 \end{pmatrix}, \ldots, \begin{pmatrix} 0 & 0 \\ 0 & \mathbf{L}_{\langle ee\rangle,2} \end{pmatrix}\right\}\right). \tag{A.9}$$

On the other hand, the four operations on the two channels of the $\Lambda_B$ map can be written as

$$\begin{aligned} \Lambda_B \simeq & \underbrace{\begin{pmatrix} \mathbf{L}_{\langle nn\rangle,1} & 0 \\ 0 & \mathbf{L}_{\langle ee\rangle,2} \end{pmatrix}}_{\text{identity}} + \underbrace{\begin{pmatrix} \mathbf{L}_{\langle nn\rangle,1} & \mathbf{L}_{\langle ne\rangle,1} \\ 0 & 0 \end{pmatrix}}_{\text{pool/broadcast edges}} \\ & + \underbrace{\begin{pmatrix} 0 & 0 \\ \mathbf{L}_{\langle en\rangle,1} & \mathbf{L}_{\langle ee\rangle,2} \end{pmatrix}}_{\text{pool/broadcast nodes}} + \underbrace{\begin{pmatrix} \mathbf{L}_{\langle nn\rangle,0} & \mathbf{L}_{\langle ne\rangle,0} \\ \mathbf{L}_{\langle en\rangle,0} & \mathbf{L}_{\langle ee\rangle,0} \end{pmatrix}}_{\text{pool/broadcast all}}. \end{aligned} \tag{A.10}$$

In particular,

- identity: if no pooling is applied, we simply map input node and edge features into output node and edge features, respectively. Note that, since we map two input channels into two output channels, we have four independent parameters associated with this operation, but two of them are redundant.

- pool/broadcast edges: when pooling and broadcasting over the edge dimension we are mapping input edge features into output node features, like the example in Eq. (A.3). Input node features are also mapped into (multiples of) themselves. Similarly to the previous case, two of the four parameters are redundant.

- pool/broadcast nodes: when pooling and broadcasting over the node dimension we are mapping input node features into output edge features, like the example in Eq. (A.4). Input edge features are also mapped into (multiples of) themselves. Similarly to the previous case, two of the four parameters are redundant.

- pool/broadcast all: when pooling and broadcasting over both dimensions we get the pooled node

and pooled edge features across the entire matrix, which we can interpret as either node or edge features as described in the previous paragraph. We can use the four independent parameters associated with this operation to identify it with the operators that map pooled nodes to node and edges, and pooled edges to node and edges.

From Eq. (A.10), a single node-edge $\Lambda_B$ layer does not generate $\mathbf{L}_{\langle ee\rangle,1}$, which corresponds to pooling the original matrix $\mathbf{A}$ (ignoring the diagonal) to its side and rebroadcasting that to the full matrix (in order to preserve symmetry, the pooled one-dimensional tensor has to be rebroadcasted both across rows and columns). Thus, the subspace $V_{\Lambda_B} \subseteq R^{N_G \times N_G}$ of node and edge features spanned by a single $\Lambda_B$ map is a subspace of $V_{\Lambda_A}$,

$$V_{\Lambda_B} = V_{\Lambda_A} / \text{span}(\left\{ \begin{pmatrix} 0 & 0 \\ 0 & \mathbf{L}_{\langle ee\rangle,1} \end{pmatrix} \right\}) \subset V_{\Lambda_A}. \tag{A.11}$$

**Table A.1:** The nine operators of the $S_N$-equivariant map $\Lambda_A$ for an undirected graph. An operator $\mathbf{L}_{\langle ab\rangle,i}$ maps input $b$ features into output $a$ features (with $a$ and $b$ labeling either node or edges), and with $i$ representing the dimension of the pooled features tensor. Rows represent the pooled features, and columns targets each row is being broadcasted to. PARTIALLY-POOLED EDGES corresponds to pooling the $\mathbf{A}$ (ignoring its diagonal) either across rows or columns. POOLED NODES and POOLED EDGES corresponds to pooling over all node and edge features, respectively.

| POOLED FEATURES / BROADCAST TO | NODES | EDGES |
|---|---|---|
| EDGES | - | $\mathbf{L}_{\langle ee\rangle,2}$ |
| NODES | $\mathbf{L}_{\langle nn\rangle,1}$ | $\mathbf{L}_{\langle en\rangle,1}$ |
| PARTIALLY-POOLED EDGES | $\mathbf{L}_{\langle ne\rangle,1}$ | $\mathbf{L}_{\langle ee\rangle,1}$ |
| POOLED NODES | $\mathbf{L}_{\langle nn\rangle,0}$ | $\mathbf{L}_{\langle en\rangle,0}$ |
| POOLED EDGES | $\mathbf{L}_{\langle ne\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,0}$ |

**Table A.2:** Multiplication of operators defined in Table A.1. Here we report multiplications of the type $\mathbf{L}_{\langle an\rangle,i}\mathbf{L}_{\langle nb\rangle,j}$ with $n$ labeling nodes, and $a$ and $b$ labeling either nodes or edges. Columns should be applied first, e.g., the entry at row $\mathbf{L}_{\langle nn\rangle,0}$ and column $\mathbf{L}_{\langle nn\rangle,1}$ means $\mathbf{L}_{\langle nn\rangle,0}\left(\mathbf{L}_{\langle nn\rangle,1}\eta\right) \simeq \mathbf{L}_{\langle nn\rangle,0}\eta$, where $\eta$ represents node features. The symbol $\simeq$ indicates that all operators are defined up to a multiplicative constant.

| | $\mathbf{L}_{\langle nn\rangle,0}$ | $\mathbf{L}_{\langle nn\rangle,1}$ | $\mathbf{L}_{\langle ne\rangle,0}$ | $\mathbf{L}_{\langle ne\rangle,1}$ |
|---|---|---|---|---|
| $\mathbf{L}_{\langle nn\rangle,0}$ | $\mathbf{L}_{\langle nn\rangle,0}$ | $\mathbf{L}_{\langle nn\rangle,0}$ | $\mathbf{L}_{\langle ne\rangle,0}$ | $\mathbf{L}_{\langle ne\rangle,0}$ |
| $\mathbf{L}_{\langle nn\rangle,1}$ | $\mathbf{L}_{\langle nn\rangle,0}$ | $\mathbf{L}_{\langle nn\rangle,1}$ | $\mathbf{L}_{\langle ne\rangle,0}$ | $\mathbf{L}_{\langle ne\rangle,1}$ |
| $\mathbf{L}_{\langle en\rangle,0}$ | $\mathbf{L}_{\langle en\rangle,0}$ | $\mathbf{L}_{\langle en\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,0}$ |
| $\mathbf{L}_{\langle en\rangle,1}$ | $\mathbf{L}_{\langle en\rangle,0}$ | $\mathbf{L}_{\langle en\rangle,1}$ | $\mathbf{L}_{\langle ee\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,1}$ |

**Table A.3:** Same as in Table A.2 but for multiplications of the type $\mathbf{L}_{\langle ae\rangle,i}\mathbf{L}_{\langle eb\rangle,j}$ with $e$ labeling edges, and $a$ and $b$ labeling either nodes or edges.

| | $\mathbf{L}_{\langle en\rangle,0}$ | $\mathbf{L}_{\langle en\rangle,1}$ | $\mathbf{L}_{\langle ee\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,1}$ | $\mathbf{L}_{\langle ee\rangle,2}$ |
|---|---|---|---|---|---|
| $\mathbf{L}_{\langle ne\rangle,0}$ | $\mathbf{L}_{\langle nn\rangle,0}$ | $\mathbf{L}_{\langle nn\rangle,0}$ | $\mathbf{L}_{\langle ne\rangle,0}$ | $\mathbf{L}_{\langle ne\rangle,0}$ | $\mathbf{L}_{\langle ne\rangle,0}$ |
| $\mathbf{L}_{\langle ne\rangle,1}$ | $\mathbf{L}_{\langle nn\rangle,0}$ | $\mathbf{L}_{\langle nn\rangle,0}+\mathbf{L}_{\langle nn\rangle,1}$ | $\mathbf{L}_{\langle ne\rangle,0}$ | $\mathbf{L}_{\langle ne\rangle,0}+\mathbf{L}_{\langle ne\rangle,1}$ | $\mathbf{L}_{\langle ne\rangle,1}$ |
| $\mathbf{L}_{\langle ee\rangle,0}$ | $\mathbf{L}_{\langle en\rangle,0}$ | $\mathbf{L}_{\langle en\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,0}$ |
| $\mathbf{L}_{\langle ee\rangle,1}$ | $\mathbf{L}_{\langle en\rangle,0}$ | $\mathbf{L}_{\langle en\rangle,0}+\mathbf{L}_{\langle en\rangle,1}$ | $\mathbf{L}_{\langle ee\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,0}+\mathbf{L}_{\langle ee\rangle,1}$ | $\mathbf{L}_{\langle ee\rangle,1}$ |
| $\mathbf{L}_{\langle ee\rangle,2}$ | $\mathbf{L}_{\langle en\rangle,0}$ | $\mathbf{L}_{\langle en\rangle,1}$ | $\mathbf{L}_{\langle ee\rangle,0}$ | $\mathbf{L}_{\langle ee\rangle,1}$ | $\mathbf{L}_{\langle ee\rangle,2}$ |

*Proof of statement (b).* Table A.2 and Table A.3 collect results for the composition of operators defined in Table A.1. It is straightforward to derive these results from the definition of the operators. As an example, consider the composition of the two maps $\mathbf{L}_{\langle ne\rangle,1}\mathbf{L}_{\langle en\rangle,1}$. Here, $\mathbf{L}_{\langle en\rangle,1}$ is the operator that broadcasts node features to edge features, it corresponds to taking the diagonal of the $\mathbf{A}$ matrix, broadcasting it across rows and columns, and summing the results (to restore symmetry). Its domain is node features $\eta$ and its range is edge features $\varepsilon$. On the other hand, $\mathbf{L}_{\langle ne\rangle,1}$ pools edge features to one dimension and broadcasts them to nodes. It corresponds to pooling the $\mathbf{A}$ matrix (ignoring the diagonal) across either rows or columns and broadcasting the result to the diagonal of $\mathbf{A}$. This composition has an alternative expression as $\mathbf{L}_{\langle ne\rangle,1}\mathbf{L}_{\langle en\rangle,1} \simeq (\mathbf{L}_{\langle nn\rangle,0}+\mathbf{L}_{\langle nn\rangle,1})$. Let $\eta$ be the vector of node features, then we get

$$\begin{aligned}\mathbf{L}_{\langle ne\rangle,1}\left(\mathbf{L}_{\langle en\rangle,1}(\eta_i)\right) &= \mathbf{L}_{\langle ne\rangle,1}(\mathrm{vec}(\eta_i+\eta_j)) \\ &= \sum_{\substack{j=1\\ j\neq i}}^{N}(\eta_i+\eta_j) = \sum_{j=1}^{N}\eta_j+(N-2)\eta_i \\ &\simeq \mathbf{L}_{\langle nn\rangle,0}(\eta_i)+\mathbf{L}_{\langle nn\rangle,1}(\eta_i),\end{aligned} \tag{A.12}$$

where $\mathbf{L}_{\langle nn\rangle,0}$ is the operator that pools node features and rebroadcast them to nodes, and $\mathbf{L}_{\langle nn\rangle,1}$ is an identity operator that rebroadcasts node features into node features. As in the previous section, $\simeq$ indicates that operators are defined up to a multiplicative constant. All results in Table A.2 and Table A.3 can be derived in a similar way. Using these multiplication rules, we compose two $\Lambda_B$ maps and find

$$\begin{aligned}\Lambda_B^2 &\simeq \begin{pmatrix}\mathbf{L}_{\langle nn\rangle,0}+\mathbf{L}_{\langle nn\rangle,1} & \mathbf{L}_{\langle ne\rangle,0}+\mathbf{L}_{\langle ne\rangle,1}\\ \mathbf{L}_{\langle en\rangle,0}+\mathbf{L}_{\langle en\rangle,1} & \mathbf{L}_{\langle ee\rangle,0}+\mathbf{L}_{\langle ee\rangle,1}+\mathbf{L}_{\langle ee\rangle,2}\end{pmatrix} \\ &\simeq \Lambda_A \implies V_{\Lambda_B^2}=V_{\Lambda_A}.\end{aligned} \tag{A.13}$$

Two stacked $\Lambda_B$ maps span the same subspace of operators and thus output features as a single $\Lambda_A$ map. Note from Table A.2 that $\mathbf{L}_{\langle ee\rangle,1}$ missing from a single $\Lambda_B$ is generated by composing $\mathbf{L}_{\langle ee\rangle,1} \simeq \mathbf{L}_{\langle en\rangle,1}\mathbf{L}_{\langle ne\rangle,1}$, that is edge features are pooled to one dimension and broadcasted to nodes in the first layer

through $\mathbf{L}_{\langle ne \rangle,1}$, and then re-broadcasted across rows and columns, like all the other node features, in the second layer through $\mathbf{L}_{\langle ne \rangle,1}$. Furthermore, using the same multiplication rules, we find that

$$\Lambda_A \simeq \Lambda_A^m \simeq \Lambda_B^{1+m} \quad \forall m \in N \,|\, m \geq 1, \tag{A.14}$$

thus by taking a single $\Lambda_A$ map or by stacking two $\Lambda_B$ maps we span a maximal node and edge feature space. □