Equivariant Heterogeneous Graph Neural Networks

Daniel Levy

Computer Science
McGill University, Montreal

August 2022

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

Abstract

Many real-world datasets include multiple distinct types of objects and relations, and so they are naturally best represented by heterogeneous graphs. However, the most common forms of neural networks operating on graphs either assume that their input graphs are homogeneous, or they flatten heterogeneous graphs into homogeneous ones, losing valuable information in the process. Any neural network that acts on graph data should be equivariant or invariant to permutations of nodes, but this is complicated when there are multiple distinct node and edge types. This thesis presents graph neural networks that are composed of linear layers that are maximally expressive while being equivariant only to permutations of nodes within each node type. Their effectiveness on heterogeneous graph node classification and link prediction benchmarks is shown, and synthetic experiments are conducted to demonstrate how the networks respond to different datasets. Lastly, this technique is extended to higher-order relations.

Résumé

De nombreux ensembles de données du monde réel comprennent plusieurs types distincts d'objets et de relations, et sont donc naturellement mieux représentés par des graphes hétérogènes. Cependant, les formes les plus courantes de réseaux neuronaux opérent sur des graphes supposent que les graphes d'entrée sont homogènes ou aplatissent les graphes hétérogènes en graphes homogènes, perdant ainsi des informations précieuses. Tout réseau neuronal qui agit sur des données de graphe doivent être équivariant ou invariant aux permutations de nœuds, mais cela se complique lorsqu'il existe plusieurs types distincts de nœuds et d'arêtes. Cette thèse présente des réseaux neuronaux de graphes composés de couches linéaires qui sont d'une expressivité maximale tout en étant équivariants uniquement aux permutations de nœuds dans chaque type. Leur efficacité sur des benchmarks hétérogènes de classification de nœuds de graphes et de prédiction de liens est démontrée, et des expériences synthétiques sont menées pour montrer comment les réseaux répondent à différents jeux de données. Enfin, nous montrons comment étendre cette technique aux relations d'ordre supérieur.

Acknowledgements

I would firstly like to express my sincere gratitude to my supervisor, Dr. Siamak Ravanbakhsh, for all the help and terrific guidance he has provided throughout my Masters studies. I would also like thank the members of our research group – Tara, Mehran, Hugo, Arnab, Vineet, Oumar, and Christopher – for their support, for fostering a great working environment, and for the many great conversations we've had about our research. Lastly, I am incredibly thankful for all my friends and family who have supported me over the past two very unusual years.

Contents

List of Tables

List of Figures

1

Introduction

Many real-world datasets and problems can be modelled as sets of objects with different relationships between them, and so graphs are a natural choice for representing these problems. Common examples include modelling interactions between users in a social network, predicting properties of molecules, or modelling connections between entities in a knowledge base.

Graph neural networks (GNNs) have become a popular technique for node and graph-level property predictions. These models have mostly focused on standard homogeneous graphs, wherein all nodes and edges are treated the same, with any differences encoded as feature vectors. However, in practical application settings, data is often complex and multi-typed, necessitating the use of heterogeneous graphs, where nodes and edges can be of different types with potentially completely different semantics. Typical ways to apply GNNs to heterogeneous networks involve preprocessing techniques such as encoding node and edge types into feature vectors, or collapsing heterogeneous networks into homogeneous ones by replacing paths along multiple different edge types with single edges. These techniques reduce the structural information available for any network to learn from and often require domain knowledge and hand-engineered features.

In this thesis, we design neural network architectures that can operate directly on entire heterogeneous graphs while fully respecting the independence and relationships between different node and edge types. We model a heterogeneous graph as a collection of node-node

adjacency matrices, one for each edge type, and create mappings from each edge type to every other edge type. For example, in a heterogeneous network that includes PUBLICATIONS, AUTHORS, and VENUES, and the relationships between these entities, our model can learn how PUBLICATION-published-at-VENUE relationships may influence AUTHOR-associated-with-VENUE relationships by constructing a linear mapping between their adjacency matrices; see Fig. 1.1.

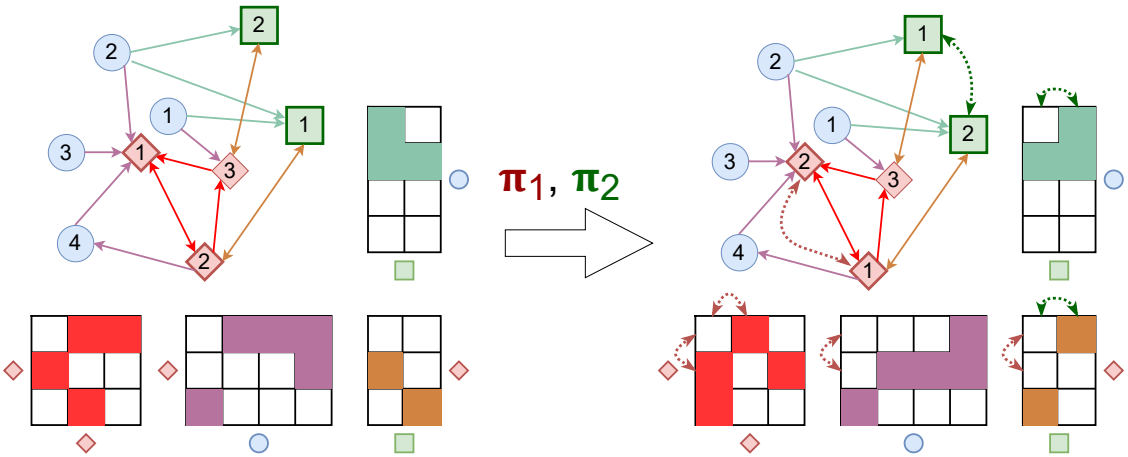


Figure 1.1: Left: An example heterogeneous graph, with 3 node types and 4 edge types. ◇ represents AUTHORS, ○ represents PUBLICATIONS, and □ represents VENUES. The graph and its adjacency matrices are shown.
Right: The effect of applying separate permutations π_1 and π_2 to the AUTHOR and VENUE nodes. While the graph itself is unaffected, with nodes simply relabelled, the adjacency matrices are modified.

A key property of any neural network that operates on graphs is that they must be *invariant* or *equivariant* to permutations of nodes. That is to say, if a graph with N nodes is represented by an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, for any permutation matrix $\pi \in \{0,1\}^{N \times N}$, a neural network $f : \mathbb{R}^{N \times N} \to \mathbb{R}^{N \times N}$ must have the property that $f(\pi \boldsymbol{A} \pi^\top) = \pi f(\boldsymbol{A}) \pi^\top$ (equivariance), or when making graph-level predictions with $f : \mathbb{R}^{N \times N} \to \mathbb{R}$, we require invariance $f(\pi \boldsymbol{A} \pi^\top) = f(\boldsymbol{A})$. For heterogeneous graphs, this invariance or equivariance constraint is to permutations within each node type.

In this thesis, we identify all linear operations that map one adjacency matrix to another while maintaining permutation equivariance within each separate node type. By combining these operations, we are able to construct maximally expressive linear equivariant layers that can then be stacked together to produce a heterogeneous graph neural network. We

create two different architectures, and apply them to two common heterogeneous graph tasks: node classification, and link prediction. We evaluate the results on a standard heterogeneous graph benchmark to determine practical usefulness, as well as a set of synthetic datasets to establish the relationship between dataset features and architecture performance. Finally, we extend our treatment to the general case of relationships involving hyperedges between more than two node types, providing a general prescription of how to efficiently implement linear layers that act on heterogeneous hypergraphs.

Chapter 2 gives an overview of graph learning, with an emphasis on graph neural networks and on heterogeneous graphs, highlighting the need for more expressive and versatile models. Chapter 3 introduces the concepts of equivariance and invariance as motivators for neural network design, and we derive neural network layers that are invariant and equivariant to permutations in heterogeneous graphs. Chapter 4 demonstrates practical architectural implementations using the layer, and applies them to real and synthetic datasets. Extensions to hypergraphs are shown in Chapter 5, and a discussion of results is included in 6.

All work in each chapter presented in this thesis is original work by the author, with support from the author's advisor, Dr. Siamak Ravanbakhsh. A summary of the notation used in this thesis is included in an appendix, Table A.1.

2

Graph Learning

A graph $\mathcal{G}$ can be defined as a tuple $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ where each $v \in \mathcal{V}$ is a node, potentially with attributes, and where $e \in \mathcal{E}$ is an edge between two nodes, also potentially with its own attributes. The set of neighbours (nodes connected by an edge) of a node v is denoted by $\mathcal{N}(v)$, and the number of neighbours is the node's degree, $\deg(v) = |\mathcal{N}(v)|$. The k-hop neighbours of a node v are the nodes reachable by following a path along k edges or fewer. One common representation of the graph is with an adjacency matrix $\boldsymbol{A} \in \{0,1\}^{N \times N}$, where $N = |\mathcal{V}|$, and where $\boldsymbol{A}_{i,j} = 1$ implies that there exists an edge between v_i and v_j. Graphs maybe undirected, in which case an edge from node v_i to node v_j is the same as one from v_j to v_i and so $\boldsymbol{A}_{i,j} = \boldsymbol{A}_{j,i}$, or directed, in which an edge from v_i to v_j does not imply an edge from v_j to v_i. Each node may v_i be accompanied by a F_v-dimensional vector of features $\boldsymbol{x}_i$, and these node features can be concatenated together to form a matrix $\boldsymbol{X}^v \in \mathbb{R}^{N \times F_v}$. Likewise, F_e-dimensional edge features can be encoded as a tensor $\mathbf{X}^e \in \mathbb{R}^{N \times N \times f_e}$. An alternative more compact representation is to encode all of this into a single data tensor $\mathbf{X} \in \mathbb{R}^{N \times N \times F}$ where $F = 1 + F_v + F_e$. The adjacency matrix is encoded with $\mathbf{X}_{:,:,1} = \boldsymbol{A}$, the node data on diagonals with $\mathbf{X}_{i,i,1:F_v+1} = \boldsymbol{X}^v_i$, and the edge data on off-diagonals with $\mathbf{X}_{:,:,F_v+1:F} = \mathbf{X}^e$.

Due to the unstructured nature of many real-world datasets, graphs are useful as means of representing them. Many datasets have somewhat obvious network-like structures, such as academic citation networks, social networks, and biological networks (Barabási and Bonabeau 2003; Zitnik and Leskovec 2017). Physical objects such as molecules and proteins may be rep-

resented as graphs, with their constituent elements (atoms and amino acids, respectively) as nodes and their bonds represented by edges (Wu, Ramsundar, et al. 2018). Three-dimensional objects such as chairs or humans may be modeled using polygonal meshes or as point clouds, which both may be encoded as graphs (Simonovsky and Komodakis 2017). Knowledge about the world can be represented as a graph in a knowledge graph, where nodes represent real-world objects, and edges between them are relationships: for example, "Elvis Presley" may be a node, "United States of America" another node, and "Born in" an edge type connecting the two nodes (M. Nickel et al. 2015).

The structures of graphs may vary wildly, and so a number of measures may be used to help describe these variances. Graphs may vary in their density, i.e. the ratio of existing edges to possible edges. Graphs may be connected, meaning that any two nodes may be joined by a path, or they may be disconnected and made up of multiple separate connected components. The distribution of degrees of each node has a big impact on the properties of a network. For example, the network describing airport connections in the United States will have nodes with degrees distributed in a power-law distribution with several very-well connected hubs and many smaller airports with few connections. In contrast, the network of highways in the United States has nodes with a Poisson distribution, as each city can only connect to a limited number of highways (Barabási and Bonabeau 2003). Graphs may be homophilic, where nodes tend to be connected to other nodes with similar properties: for example, in a citation network, academic papers are likely to cite other papers in the same field. They may also be heterophilic, where dissimilar nodes are preferentially attached: for example, a network of matches on a dating website might be heterophilic when it comes to user's genders (J. Zhu, Y. Yan, et al. 2020).

The definition of a graph defined above can be extended further in a number of ways. Spatio-temporal graphs have features that may change over time, which are very useful for modelling traffic networks (Yu, Yin, and Z. Zhu 2017) or moving objects such as skeletons (S. Yan, Xiong, and Lin 2018). Heterogeneous graphs, of particular interest to this thesis, are graphs in which nodes, edges, or both may take on different types. Heterogeneous graphs

are particularly useful when different node types imply completely different types of objects, and different edge types imply different relationships with very different semantic meanings. They can be thought of as coupling a graph with mapping functions, one that maps nodes to node types, and one that maps edges to edge types. Another way to think of a heterogeneous graph is as the union of several graphs, one for each edge type, with nodes shared between them. An example of a heterogeneous graph displayed with this perspective is shown in Fig. 1.1.

Hypergraphs are another extension to graphs, in which an edge can connect more than just two nodes. If the order of the edges is fixed (e.g. if all edges involve three nodes then it is a order-3 hypergraph), then an order k hypergraph can be represented by a tensor $\mathbf{A} \in \mathbb{R}^{N^k}$. When edges can have an arbitrary order, a hypergraph can be represented by an edge-node incidence matrix $\boldsymbol{A} \in \mathbb{R}^{N \times M}$ where M is the number of edges.

2.1 Graph Neural Networks

Over the past two decades, deep learning methods have emerged as front-runners in a variety of machine learning tasks in numerous problem domains. Deep learning methods use multi-layered neural networks to encode data into vector representations, known as embeddings. These embeddings can then be used for downstream tasks, such as regression or classification. Importantly, deep learning based methods are trained in an end-to-end manner: they are able to automatically learn what features of its input data are important for it to make predictions, obviating the need for hand-engineered features.

Deep learning tasks can be supervised, in which case labels for each data point in the training set are known, and they are used to train the neural network to predict these labels. They may be semi-supervised, where only some labels are known (common in real-world datasets). They may instead be unsupervised, where no labels are provided at all, and the neural network is tasked with learning an internal representation of the data that may be used to model its distribution. Tasks may be transductive, in which case the goal is to make

predictions about data seen in the training set, or they may be inductive, where predictions are made on entirely unseen data, and requiring any solution to learn generalizable rules.

Deep learning methods have been extended to apply to graph data, creating a class of models known as graph neural networks, or GNNs. This approach was first introduced by Gori, Monfardini, and Scarselli 2005 and Scarselli et al. 2008. In contrast to most previous non-deep learning algorithms on graphs, GNNs are broadly generalizeable, applicable to a wider variety of tasks, and avoid the need to hand-engineer graph features. Furthermore, GNNs can be usually applied to inductive settings, as a trained GNN can make predictions on entirely unseen nodes or graphs.

While the range of possible applications for graph neural networks are limitless, they often fall into three broad categories: node-level predictions, link-level predictions, and graph-level predictions. Examples of node-level predictions include predicting fraudulent users in a social network, or predicting the subject of a publication in a citation network. Examples of link-prediction include predicting potential matches between users in a social network, or predicting possible missing facts in a knowledge graph. Examples of graph-level prediction include predicting chemical properties of molecules, or classifying proteins (Zhou et al. 2020).

2.1.1 Message Passing Framework

Throughout the development of graph neural networks, most methods can be thought of as belonging to a broader class of so-called message passing neural networks (Gilmer et al. 2017). In this framework, information about nodes (in the form of vector embeddings) are combined with information from their neighbours to form "messages", which are then aggregated and used to update the embeddings for the nodes. By repeating this process multiple times, node information is propagated throughout the graph along its structure. Using the formulation from Bronstein et al. 2021, we can compute a message between nodes v_i and v_j:

$$\boldsymbol{m}_{i,j} = \psi(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.1}$$

Here ψ is a differentiable and possibly learnable function, such as a multi-layered perceptron (MLP). The messages are then used to update each node:

$$\boldsymbol{z}_i = \phi\left(\boldsymbol{x}_i, \bigoplus_{j \in \mathcal{N}(v_i)} \boldsymbol{m}_{i,j}\right) \tag{2.2}$$

Here, ϕ is another differentiable and potentially learnable function, and $\bigoplus$ is an aggregation function that does not depend on the order of its inputs, such as addition, or taking the mean, or taking the max. The updated node features $\boldsymbol{z}_i$ can then be used as inputs to another round of message passing.

The resulting node embedding vectors may then be used to perform node-level prediction tasks. To accomplish edge-level prediction tasks, the embeddings of two candidate nodes may be compared together (for example, by taking their dot product), and the result can be used to make predictions about a potential edge between the nodes. To perform graph-level predictions, the node-level embeddings of a graph may be aggregated in some way.

A commonly used type of neural network architecture that fits into the message-passing framework is the graph convolution network, or GCN (Kipf and Welling 2016). In a GCN, at each layer a message is computed as $\boldsymbol{m}_{i,j} = c_{i,j}\boldsymbol{x}_i$ where $c_{i,j} = (\deg(v_i)\deg(v_j))^{-1/2}\,\boldsymbol{A}_{i,j}$. This message is plugged into Equation 2.2 using a summation as the aggregation function $\bigoplus$, computing $\boldsymbol{z}_i = \sigma\left(\boldsymbol{W}\sum_{j\in\mathcal{N}(v_i)}\boldsymbol{m}_{i,j}\right)$, where $\boldsymbol{W}$ is a learned parameter matrix, and σ is a nonlinear function, such as the ReLU function.

Another common neural network architecture that fits into this framework is the graph attention network, or GAT (Veličković et al. 2017). Rather than using a simple scalar $c_{i,j}$ to compute the influence of one node on its neighbour, it instead uses an attention mechanism $a(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to calculate a weight dependent on the features of each node. The attention mechanism is defined as $a(\boldsymbol{x}_i, \boldsymbol{x}_j) = \text{softmax}\left(\boldsymbol{a}^\intercal[\boldsymbol{W}\boldsymbol{x}_i||\boldsymbol{W}\boldsymbol{x}_j]\right)$ where $\boldsymbol{a}$ is a learned parameter vector, and $||$ is the concatenation operation. It is then used to compute the message $\boldsymbol{m}_{i,j} = a(\boldsymbol{x}_i, \boldsymbol{x}_j)\boldsymbol{W}\boldsymbol{x}_i$. This message is plugged into Equation 2.2, computing $\boldsymbol{z}_i = \sigma\left(\sum_{j\in\mathcal{N}(v_i)}\boldsymbol{m}_{i,j}\right)$. The GAT may be multiheaded, where $\boldsymbol{z}_i$ becomes the concatenation of H vectors $\boldsymbol{z}_i^h$, each computed with their own $\boldsymbol{W}^h$ and $\boldsymbol{a}^h$.

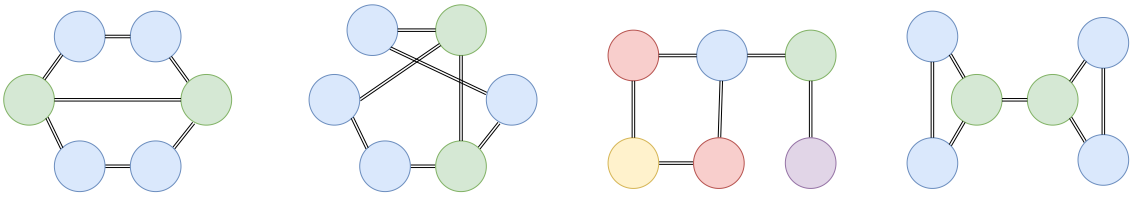
This general concept of graph neural networks actually subsumes several other common types of neural networks. A typical convolutional neural network (CNN) (LeCun, Bottou, et al. 1998) for image data can be thought of as a type of graph convolution network applied to a two-dimensional grid-structured graph, where each node is a pixel, and edge weights are determined by the relative orientation of the pixels. The transformer network (Vaswani et al. 2017), which has found great success in natural language processing applications, can be thought of as a type of GAT applied to graphs where each node is a word in some text, and each word is connected to every other word in that text (Kreuzer et al. 2021; Dwivedi and Bresson 2020).

2.1.2 Weisfeiler-Leman Test

The structure of message-passing neural networks closely resembles a known algorithm within graph theory, the Weisfeiler-Leman (WL) algorithm (Weisfeiler and Leman 1968). This algorithm is applied to two graphs and used as a test for whether they are isomorphic to each other: that is, whether they are identical up to a relabelling of nodes. Determining whether two graphs are isomorphic to each other is, in general, not known to be solvable in polynomial time.

The WL algorithm initially assigns colors c_i^0 to each node v_i in a graph $\mathcal{G}$, either uniformly, or so that nodes with unique features have unique colors. At each step $t > 0$, each node v_i counts the number of its neighbours that have each color, forming a multiset of colors $C^{t-1}(v_i) = \{\!\{c_j^{t-1} | j \in \mathcal{N}(v_i)\}\!\}$, where $\{\!\{\}\!\}$ denotes a multiset. The node then uses these features as well as its own color, c_i^{t-1}, to assign itself a new color $c_i^t = \phi(C^{t-1}(v_i), c_i^{t-1})$, where ϕ is an injective function, like a perfect hash function. We can then count all the colors of nodes in the graph: $C^t(\mathcal{G}) = \{\!\{c_i^t | i \in \mathcal{V}\}\!\}$. If we apply the WL algorithm to graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ in parallel (using the same function ϕ and same node color initialization procedure), then if at any time t, $C^t(\mathcal{G}_1) \neq C^t(\mathcal{G}_2)$, we can declare that $\mathcal{G}_1$ and $\mathcal{G}_2$ are not isomorphic to each other. If not, we take another step in the algorithm. If the algorithm converges on both graphs (i.e. no node colors change for one step), then we do not know

whether they are isomorphic or not. While the WL test can often distinguish non-isomorphic graphs, it fails in some very obvious ways. An example is shown in Fig. 2.1.



(a) (b) (c) (d)

Figure 2.1: Four graphs, shown with stable colorings resulting from running the WL algorithm. Graphs (a) and (b) are isomorphic, and are assigned the same colorings. Graph (c) is not isomorphic to (a), and the WL test correctly distinguishes them. However, the WL test cannot distinguish graph (a) from the non-isomorphic graph (d).

The GNN message passing step of Eq. (2.2) is equivalent to one round of color-updating in the WL algorithm, if ϕ and $\bigoplus$ are both injective functions. This equivalance means that GNNs that follow a message passing framework necessarily inherit the same shortcomings as the WL test: they are not able to distinguish graphs that would not be distinguishable by the WL test (Morris, Ritzert, et al. 2019; Xu et al. 2018).

By defining an upper limit on the distinguishing power of message-passing GNNs, this equivalence has motivated new, more powerful graph architectures that can surpass the WL test. The WL test can be extended by considering it to be the 1-dimensional case of a more general k-dimensional WL test, which colors k-tuples of nodes. For $k \geq 2$, the $(k+1)$-WL test is able to distinguish graphs that the k-WL test would be unable to distinguish (Cai, Fürer, and Immerman 1992). The k-WL test provides a principled way of upper-bounding the expressive power of graph neural networks (Geerts and Reutter 2022). Neural networks specifically designed to surpass the 1-WL and attain the separating powers of the k-WL test include Maron, Ben-Hamu, Serviansky, et al. 2019 and Morris, Ritzert, et al. 2019.

For a thorough review of the connections between the WL-test and graph learning, see Morris, Lipman, et al. 2021.

2.2 Heterogeneous Graph Learning

The ubiquity of complex multi-typed data in real-world problems has caused heterogeneous graph learning to attract a lot of attention in applied settings. Heterogeneous graph networks have been applied to such diverse tasks as text classification (Linmei et al. 2019), disease diagnosis (Z. Wang et al. 2021), and malicious account detection (Liu et al. 2018).

The majority of heterogeneous graph learning techniques rely on *meta-paths*: sequences of different node and edge types (Sun and Han 2012; Shi et al. 2017). For example, in a citation network with AUTHORS, PAPERS, AND VENUES, the "path" AUTHOR – PUBLICATION – VENUE – PUBLICATION – AUTHOR represents one meta-path between two AUTHORS that have published at the same venue. These meta-paths are usually hand-designed, requiring domain knowledge.

Heterogeneous graph learning techniques can be broadly classified into either "shallow" embedding models, or "deep" neural models (Dong, Hu, et al. 2020; Yang et al. 2020). Shallow methods (such as Dong, Chawla, and Swami 2017; Tang, Qu, and Mei 2015; T.-y. Fu, Lee, and Lei 2017) aggregate node attributes using techniques such as random walks over different edge types, in order to obtain structure-preserving embeddings for each node, which are then passed on to other machine learning models for downstream tasks. These are limited to transductive settings.

Deep methods extend conventional GNNs, but learn parameters or embeddings specific to each node or edge type; see Wu, Pan, et al. 2020 for a survey of homogeneous GNNs. Examples include R-GCN (Schlichtkrull et al. 2018) which extends GCN by learning edge-specific weight matrices, Heterogeneous Graph Attention Network (HAN) (X. Wang, Ji, et al. 2019) and Metapath Aggregated Graph Neural Network (MAGNN) (X. Fu et al. 2020), which extend graph attention to attend over different meta-paths. Some methods, such as Heterogeneous Graph Transformer (HGT) Hu et al. 2020 and Graph Transformer Network (GTN) (Yun et al. 2019) can automatically discover what meta-paths are worth using, but even then they are not able to capture as much information as if they were to directly use

the full heterogeneous graph. For two recent surveys of heterogeneous graph representation learning techniques, see Yang et al. 2020 and Dong, Hu, et al. 2020.

Lv et al. 2021 recently called into question whether most heterogeneous graphs neural networks are able to properly exploit the information provided by node and edge types. They show that under fair comparisons, they are often outperformed by conventional graph neural networks that simply ignore node and edge type information, such as GCN (Kipf and Welling 2016) and GAT (Veličković et al. 2017). This shortcoming motivates us to design neural networks that treat edge and node types as first-class objects, with the ability to learn the relationships between one edge type and another, and to adapt to the different semantics conveyed by different node and edge types.

3

Equivariant Heterogeneous Graph Layers

The goal of this chapter is to derive a maximally expressive layer of a neural network that operates on heterogeneous graphs, composed only of linear operations. We desire linear layers as they serve as the building block of deep learning architectures: for example, by simply alternating linear layers with nonlinear activation functions, an analogue of the multilayer perceptron can be constructed for heterogeneous graphs. As we will see, the structure of heterogeneous graphs restricts the form that these layers may have. Furthermore, the structure of these layers imply an efficient decomposition into a set of pooling and broadcasting operations.

3.1 Equivariant and Invariant Learning

When defining a graph using an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, we are forced to pick an ordering of the nodes in the graph. This ordering is usually entirely arbitrary, and we therefore do not want it to have an effect on the final results of any graph learning task. We can permute elements in $\mathbf{A}$ by multiplying it on both sides with a permutation matrix $\pi \in \{0,1\}^{N \times N}$ where $\forall i, \sum_j \pi_{i,j} = 1$ and $\forall j, \sum_i \pi_{i,j} = 1$. Using the language of group theory, a permutation matrix π is a representation of an element of the symmetric group $S(N)$, a group that contains all $N!$ permutations of a set of N nodes. We can say that permutation of node orderings is a symmetry of our data. If we are making a graph-level prediction, then

this means we want our prediction to be invariant to permutations of node orderings, i.e. $f(\boldsymbol{A}) = f(\pi \boldsymbol{A} \pi^\intercal)$. If we are making node-level or edge-level predictions, then a reordering of nodes on the input should result in the exact same reordering of the resulting node and edge predictions. In this case, we would like to be equivariant to permutations of node orderings, i.e. $\pi f(\boldsymbol{A}) \pi^\intercal = f(\pi \boldsymbol{A} \pi^\intercal)$. Without this invariance or equivariance requirement, a neural network would need to separately train on each possible permutation of each graph in its training data in order to have it robustly make predictions, with computational requirements scaling with the factorial of the size of the graph. All of the graph neural networks described in 2.1.1 are permutation equivariant or invariant.

Invariance and equivariance can be described more generally for other types of symmetries. Let's say we have a symmetry group G where elements $g \in G$ are represented with a map $\rho : G \to \mathbb{R}^{N \times N}$, a set of possible data $\mathcal{X}$, and elements of the group can act on data $x \in \mathcal{X}$ via matrix multiplication, $p(g)x$. In our previous example, $\mathcal{G}$ was the symmetric group $S(N)$ and ρ gave us $N \times N$ permutation matrices. A function $f : \mathcal{X} \to \mathcal{X}$ is said to be invariant iff $f(x) = f(\rho(g)x)$, and it is said to be equivariant iff $\rho(g)f(x) = f(\rho(g)x)$.

The invariance and equivariance requirements has proved to be a useful tool for deriving new neural networks in a principled way, an approach that has been called the Geometric Deep Learning Blueprint, as explained by Bronstein et al. 2021. Convolutional neural networks (CNNs) (LeCun, Boser, et al. 1989) were a great breakthrough for neural networks applied to image data, and they are successful because they have translational equivariance built in to them, by using a convolution kernel that is applied to a neighbourhood around each pixel. Further developments have extended this equivariance to symmetry groups such as rotations on a sphere (T. S. Cohen et al. 2018) and surfaces of manifolds (Masci et al. 2015; Monti et al. 2017).

The Deep Sets model (Zaheer et al. 2017) was one of the first deep learning models to be expressly designed to be invariant or equivariant to permutations of sets, the group $S(N)$. It does so by using just two learnable parameters per layer: one for each individual element of the set, and one for the aggregation of all elements of the set. The form of a layer of the

network resembles: $f(\boldsymbol{x}_i) = \sigma\left(w_1\boldsymbol{x}_i + w_2 \bigoplus_j \boldsymbol{x}_j\right)$, where w_1 and w_2 are learned parameters and σ is a nonlinear function. If $\bigoplus$ is set to summation, this can also be thought of an MLP layer $f(\boldsymbol{X}) = \sigma(\boldsymbol{W}\boldsymbol{X})$ where the weights are tied together, such that $\boldsymbol{W} = w_1\boldsymbol{I} + w_2(\mathbf{1}\mathbf{1}^\intercal)$ where $\mathbf{1}$ is a vector of all ones and $\boldsymbol{I}$ is an identity matrix. Indeed, Ravanbakhsh, Schneider, and Poczos 2017 shows the equivalence between parameter sharing and equivariance for discrete symmetry groups.

Several works directly seek the set of equivariant and invariant operations with this property to use them as building blocks in graph neural networks, extending beyond just unstructured sets. Of particular relevance is the work of Kondor et al. 2018, which introduces permutation equivariant operations that can be applied to tensor representations of graphs, and Maron, Ben-Hamu, Shamir, et al. 2018, which characterizes a basis for all equivariant linear operations on tensor representations of graphs and hypergraphs with potentially different node types. So far, these works have only been implemented for the cases of homogeneous graphs, and have not been extended to the more general case of heterogeneous graphs. Furthermore, while they characterise the set of equivariant linear bases, their analysis does not give a practical algorithm, since such large matrices that form the linear bases are too large to store in memory for any large graph. While Maron, Ben-Hamu, Shamir, et al. 2018 give an efficient implementation based on matrix pooling and broadcasting for standard homogeneous graphs, and Hartford et al. 2018 give an efficient implementation for the case of a single relationship between different node types, neither provide a more general implementation for arbitrary node and edge types. Albooyeh, Bertolini, and Ravanbakhsh 2019 give a pooling and broadcasting view of operations for hyper-graphs and incidence structures of other geometric entities; however, all such structures have a single node type that is assumed exchangeable. Some other related works that have a symmetry-based approach to GNNs and other permutation-equivariant structures include Maron, Ben-Hamu, Serviansky, et al. 2019; Haan, T. Cohen, and Welling 2020; Azizian and Lelarge 2020; Graham, J. Wang, and Ravanbakhsh 2019.

3.2 Equivariant Linear Maps for Heterogeneous Graphs

3.2.1 Notation

A heterogeneous graph $\mathcal{G} = \langle \mathbb{D}, \mathbb{R}, \mathbb{V}, \mathbb{X} \rangle$ is a four tuple, where $\mathbb{D} = \{1, \ldots, D\}$ is the set of node types. For each node type $d \in \mathbb{D}$ there is a set of N_d nodes $\mathbb{V}_d = \{v_1, \ldots, v_{N_d}\}$. Each edge type r is given by a relation between nodes $\mathrm{r} = \langle \overline{r}, \underline{r} \rangle$ where $\overline{r}, \underline{r} \in \mathbb{D}$ are the pair of node types that the edge type r links between. The set of all edge types is $\mathbb{R} = \{\mathrm{r}_1, \ldots, \mathrm{r}_R\}$. The set of node adjacency matrices is $\mathbb{X} = \{\boldsymbol{X}^{\mathrm{r}} \in \mathbb{R}^{N_{\overline{r}} \times N_{\underline{r}}} \mid \mathrm{r} \in \mathbb{R}\}$, one for each edge type $\mathrm{r} \in \mathbb{R}$. When $\overline{r} = \underline{r}$, such matrices can represent both node and edge attributes using diagonal and off-diagonal elements respectively. For simplicity, node and edge attributes are initially assumed to be scalar, but later it can be shown that these may be generalized to vectors using multiple channels. In the definition above, $\mathbb{D}, \mathbb{R}$, and $\mathbb{V}$ contain the blueprint of the heterogeneous graph, while $\mathbb{X}$ contains the actual data.

3.2.2 Equivariance for Heterogeneous Graphs

Given the heterogeneous graph $\mathcal{G}$ our goal is to identify all *equivariant* linear operators that map the set of matrices $\mathbb{X} = \{\boldsymbol{X}^1, \ldots, \boldsymbol{X}^R\}$ to another set of matrices $\mathbb{Y} = \{\boldsymbol{Y}^1, \ldots, \boldsymbol{Y}^R\}$ of the same form. For this, it is sufficient to identify all such maps from one edge type to another $\mathrm{L}^{\mathrm{r} \to \mathrm{r}'} : \mathbb{R}^{N_{\overline{r}} \times N_{\underline{r}}} \to \mathbb{R}^{N_{\overline{r}'} \times N_{\underline{r}'}}$. The overall equivariant map $\mathrm{L}^{\mathbb{R} \to \mathbb{R}}$ can be built from the collection $\mathrm{L}^{\mathrm{r}_1 \to \mathrm{r}_1}, \mathrm{L}^{\mathrm{r}_1 \to \mathrm{r}_2}, \ldots, \mathrm{L}^{\mathrm{r}_1 \to \mathrm{r}_R}, \ldots, \mathrm{L}^{\mathrm{r}_R \to \mathrm{r}_R}$.

The equivariance condition on the linear operator $\mathrm{L}^{\mathrm{r} \to \mathrm{r}'}$ ensures that any permutation of the input nodes of the same type leads to the same permutation of the nodes in the output for that node type. Let $\pi_d \in S(N_d)$ be a permutation matrix acting on N_d nodes of type d.

The Equivariance constraint requires

$$\mathrm{L}^{\mathrm{r}\to\mathrm{r}'}(\pi_{\bar{r}}\boldsymbol{X}^{\mathrm{r}}\pi_{\underline{r}}^{\top}) = \pi_{\bar{r}'}\,\mathrm{L}^{\mathrm{r}\to\mathrm{r}'}(\boldsymbol{X}^{\mathrm{r}})\pi_{\underline{r}'}^{\top} \quad \forall \pi_{\bar{r}}, \pi_{\underline{r}}, \pi_{\bar{r}'}, \pi_{\underline{r}} \in S(N_{\bar{r}}) \times S(N_{\underline{r}}) \times S(N_{\bar{r}'}) \times S(N_{\underline{r}'}) \tag{3.1}$$

where the permutation matrices correspond to two pairs of node types that appear in the input (r) and output edge types (r′).

As also observed in related contexts (Kondor et al. 2018; Albooyeh, Bertolini, and Ravanbakhsh 2019) such linear operators often involve pooling and broadcasting over input and output matrices. Our plan is to enumerate all such operations and prove that these are indeed the only linear operations with the desired equivariance property Eq. (3.1).

Example 1: To build an intuition for these operations, consider two relations between ⟨AUTHOR, VENUE⟩ and ⟨PUBLICATION, AUTHOR⟩. Let $\mathrm{r} = \langle 1, 2\rangle$ denote the former and $\mathrm{r}' = \langle 3, 1\rangle$ be the latter, noting that these two edge types have a node type in common. The desired linear map $\mathrm{L}^{\mathrm{r}\to\mathrm{r}'}$ should be equivariant to independent permutation of AUTHOR nodes, PUBLICATION nodes and VENUE nodes in our graph. The results that follow this example show that any equivariant $\mathrm{L}^{\mathrm{r}\to\mathrm{r}'}$ has the following form:

$$\mathrm{L}^{\mathrm{r}\to\mathrm{r}'}(\boldsymbol{X}) = w_1(\boldsymbol{X}\mathbf{1}_{N_{\underline{r}}}\mathbf{1}_{N_{\bar{r}'}})^{\top} + w_2\mathbf{1}_{N_{\bar{r}'}}(\mathbf{1}_{N_{\bar{r}}}^{\top}\boldsymbol{X}\mathbf{1}_{N_{\underline{r}}})\mathbf{1}_{N_{\underline{r}'}}^{\top} \tag{3.2}$$

where $w_1, w_2 \in \mathbb{R}$ are arbitrary weights and $\mathbf{1}_N$ is the identity vector of length N. Here, following Zaheer et al. 2017 we are performing pooling and broadcasting operations using multiplication by identity vectors. The first operation $(\boldsymbol{X}\mathbf{1}_{N_{\underline{r}}}\mathbf{1}_{N_{\bar{r}'}})^{\top}$ pools over the columns of $\boldsymbol{X}$ (*i.e.*, VENUES), and broadcasts the resulting column vector to create a $N_{\underline{r}'} \times N_{\bar{r}'}$ matrix which is then transposed to match the dimensions of the target edge type. We can think of the pooling operation above as collecting edge attributes from all the VENUES that are adjacent to each AUTHOR. Similarly, the broadcasting operation disperses this pooled information over all the PUBLICATION nodes adjacent to each AUTHOR. This example shows that an equivariant linear map is able to propagate relevant information across different edge types.

3.2.3 Characterizing Equivariant Linear Maps

After describing the constraints implied by the structure of heterogeneous graphs, the task is now to identify all equivariant linear operations for a given pair of edge types r, r'. In addition to the pooling, broadcasting, and transpose operation used in the example above, we need one additional operation, namely diag. We overload this operation so that for a square matrix, $\mathrm{diag} : \mathbb{R}^{N \times N} \to \mathbb{R}^N$ extracts the diagonal, and for a vector input, $\mathrm{diag} : \mathbb{R}^N \to \mathbb{R}^{N \times N}$ outputs a square matrix with that vector on its diagonal – this means $\mathrm{diag}(\mathrm{diag}(\boldsymbol{x})) = \boldsymbol{x}$ and $\mathrm{diag}(\mathrm{diag}(\boldsymbol{X})) = \boldsymbol{X} \odot \boldsymbol{I}$ (where $\odot$ is the Hadamard product and $\boldsymbol{I}$ is an identity matrix).

The idea is to create all possible combinations of the linear operations above that take us from a $N_{\overline{r}} \times N_{\underline{r}}$ matrix to a $N_{\overline{r}'} \times N_{\underline{r}'}$ matrix. These operations vary based on the equality of some of these dimension – for example if $\overline{r} = \underline{r}$ then the operation $\mathrm{diag}(\boldsymbol{X})$ is well-defined, and otherwise it is not feasible. To help with this enumeration, any such linear operation can be broken into parts:

Contraction operations These include pooling over the rows, columns, both rows and columns, extraction of diagonal and pooling over the diagonal, as well as the identity operation. The result could be a scalar, a vector, or a matrix. Below, we use $z, \boldsymbol{z}$, and $\boldsymbol{Z}$ to denote these intermediate products, and identify the condition under which we can perform each of these contraction operations:

Operation		**Condition**
1. Identity operation	$\boldsymbol{Z}^{\langle \overline{r}, \underline{r} \rangle} = \boldsymbol{X}^{\langle \overline{r}, \underline{r} \rangle}$	-
2. Pooling over rows	$\boldsymbol{z}^{\overline{r}} = \boldsymbol{X}^{\langle \overline{r}, \underline{r} \rangle} \mathbf{1}_{N_{\underline{r}}}$	-
3. Pooling over columns	$\boldsymbol{z}^{\underline{r}} = \boldsymbol{X}^{\langle \overline{r}, \underline{r} \rangle^\top} \mathbf{1}_{N_{\overline{r}}}$	-
4. Pooling over rows and columns	$z = \mathbf{1}_{N_{\overline{r}}}^\top \boldsymbol{X}^{\langle \overline{r}, \underline{r} \rangle} \mathbf{1}_{N_{\underline{r}}}$	-
5. Extracting the diagonal	$\boldsymbol{z}^{\overline{r}} = \mathrm{diag}(\boldsymbol{X}^{\langle \overline{r}, \underline{r} \rangle})$	$\overline{r} = \underline{r}$
6. Pooling the diagonal	$z = \mathrm{diag}(\boldsymbol{X}^{\langle \overline{r}, \underline{r} \rangle})^\top \mathbf{1}_{N_{\overline{r}}}$	$\overline{r} = \underline{r}$

Expansion operations These operations expand the intermediate value to produce the target matrix. The operations include broadcasting over rows, columns, both rows and

columns, diagonal placement, diagonal broadcasting, as well as the identity operation and matrix transpose.

1. Identity operation	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \boldsymbol{Z}^{\langle \overline{r}, \underline{r} \rangle}$	$\overline{r}' = \overline{r}, \underline{r}' = \underline{r}$
2. Transpose	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \boldsymbol{Z}^{\langle \overline{r}, \underline{r} \rangle^\top}$	$\overline{r}' = \underline{r}, \underline{r}' = \overline{r}$
3 Broadcasting over columns	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \boldsymbol{z}^{\overline{r}} \mathbf{1}^\top_{N_{\underline{r}'}}$	$\overline{r} = \overline{r}'$
	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \boldsymbol{z}^{\underline{r}} \mathbf{1}^\top_{N_{\underline{r}'}}$	$\underline{r} = \overline{r}'$
4. Broadcasting over rows	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \mathbf{1}_{N_{\overline{r}'}} \boldsymbol{z}^{\overline{r}\top}$	$\overline{r} = \underline{r}'$
	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \mathbf{1}_{N_{\overline{r}'}} \boldsymbol{z}^{\underline{r}\top}$	$\underline{r} = \underline{r}'$
5. Broadcast over rows and cols	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \mathbf{1}_{N_{\overline{r}'}} z \mathbf{1}^\top_{N_{\underline{r}'}}$	-
6. Placing the diagonal	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \mathrm{diag}(\boldsymbol{z}^{\overline{r}})$	$\overline{r}' = \underline{r}' = \overline{r}$
	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \mathrm{diag}(\boldsymbol{z}^{\underline{r}})$	$\overline{r}' = \underline{r}' = \underline{r}$
7. Broadcasting over the diagonal	$\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \mathrm{diag}(z \mathbf{1}_{N_{\overline{r}'}})$	$\overline{r}' = \underline{r}'$

Theorem 3.2.1. *Given two edge types* r, r'*, all the linear maps* $\mathrm{L}^{\mathrm{r}\to\mathrm{r}'} : \mathbb{R}^{N_{\overline{r}} \times N_{\underline{r}}} \to \mathbb{R}^{N_{\overline{r}'} \times N_{\underline{r}'}}$ *that satisfy the equivariance condition of Eq.* (3.1) *are produced using the contraction and expansion operations above.*

Proof. It is easy to see that all expansion and contraction operations are equivariant, linear, and independent from one another. The composition of a contraction and expansion operation is also equivariant and linear. To prove that these operations represent all linear equivariant maps, we count them, and compare the counts to the results obtained by Maron, Ben-Hamu, Shamir, et al. 2018 on the number of linear independent bases for equivariant maps for graphs.

We look at all valid combinations of contraction and expansion operations. There is 1 contraction of the form $\boldsymbol{X} \to \boldsymbol{Z}$ (op. 1) , 3 of the form $\boldsymbol{X} \to \boldsymbol{z}$ (ops. 2, 3, and 5), and 2 of the form $\boldsymbol{X} \to z$ (ops. 4 and 6). There are 2 expansions of the form $\boldsymbol{Z} \to \boldsymbol{Y}$ (ops. 1 and 2), 3 expansions of the form $\boldsymbol{z} \to \boldsymbol{Y}$ (ops. 3, 4, and 6) and 2 of the form $z \to \boldsymbol{Y}$ (ops. 5 and 7). A contraction of the form $\boldsymbol{X} \to \boldsymbol{Z}$ can only be paired with an expansion of the form

$\boldsymbol{Z} \to \boldsymbol{Y}$, and so on for the other forms. This yields $(1 \times 2) + (3 \times 3) + (2 \times 2) = 15$ different operations when $\mathrm{r} = \mathrm{r}'$, matching the results of Maron, Ben-Hamu, Shamir, et al. 2018.

We can generalize this to cases where $\mathrm{r} \neq \mathrm{r}'$. First, we define the function $\kappa(\mathrm{r}, d)$ to be the number of times node type d appears in the tuple r. From Theorem 3 of Maron, Ben-Hamu, Shamir, et al. 2018, the number of possible operations between a matrix $\boldsymbol{X}^r$ and $\boldsymbol{Y}^{r'}$ is:

$$C(r, r') = \prod_{d \in \mathbb{D}} \mathrm{Bell}\left(\kappa(\mathrm{r}, d) + \kappa(\mathrm{r}', d)\right) \tag{3.3}$$

Here, $\mathrm{Bell}(k)$ is the k-th Bell number; the k-th Bell number is defined as the number of ways a set with k elements may be partitioned.

For each possible set of equalities between $\overline{r}, \underline{r}, \overline{r}'$, and $\underline{r}'$, if we count the number of valid combinations of contraction and expansion operations, it matches the count obtained by Eq. (3.3). For example, going back to Example 1, if $\mathrm{r} = \langle 1, 2 \rangle$ and $\mathrm{r}' = \langle 3, 1 \rangle$, then only condition $\overline{r} = \underline{r}'$ is met. The valid contraction operations are 1, 2, and 3, and the valid expansion operations are 4 and 5. This leads to $(1 \times 0) + (1 \times 1) + (1 \times 1) = 2$ valid combinations, matching the result predicted by Eq. (3.3) and obtaining Eq. (3.2) .

This can be manually verified for each combination of $\overline{r}, \underline{r}, \overline{r}'$, and $\underline{r}'$, but for a more general proof, see Section 5.1. □

3.3 The Neural Network Layer

Now that we have enumerated all possibilities for permutation equivariant linear mappings $\mathrm{L}^{\mathrm{r} \to \mathrm{r}'}$, we can combine them to form a linear layer that acts on a set of adjacency matrices:

$$\mathrm{L}^{\mathbb{R} \to \mathbb{R}}(\mathbb{X}) = \left\{ \sum_{\boldsymbol{X}^{\mathrm{r}} \in \mathbb{X}} \sum_{c=1}^{C(\mathrm{r}, \mathrm{r}')} w_c^{\mathrm{r} \to \mathrm{r}'} \, \mathrm{L}_c^{\mathrm{r} \to \mathrm{r}'}(\boldsymbol{X}^{\mathrm{r}}) \mid \mathrm{r}' \in \mathbb{R} \right\} \tag{3.4}$$

Here, c indexes all valid combinations of contraction and expansion operations, and $w_c^{\mathrm{r} \to \mathrm{r}'} \in \mathbb{R}$ is a weight for that combination that may be learned. In practice, both the inner and outer sum can be replaced by any permutation invariant aggregation function, such as taking the maximum or taking the mean.

3.3.1 Multiple Channels

We can extend the above definitions to include edge feature vectors in a straightforward way, if we instead replace the matrix $\boldsymbol{X}^{\mathrm{r}}$ with the tensor $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N_{\overline{r}} \times N_{\underline{r}} \times F}$ where F is some feature dimension. Now, instead of having scalar weights w_c in Eq. (3.4), we have a collection of weight matrices $\boldsymbol{W}_c \in \mathbb{R}^{F' \times F}$. Each equivariant layer can then specify the number of feature dimensions in their input and output. The weights mix between feature dimensions and not node dimensions, so the layers are still equivariant to node permutations.

3.3.2 Sparse Implementation

In practice, graphs are often very sparse, making it impractical to deal with full adjacency matrices. We instead represent a data tensor $\mathbf{X}^{\mathrm{r}}$ with M nonzero entries as a tuple $\mathrm{sp}(\mathbf{X}^{\mathrm{r}}) = \langle \boldsymbol{U}, \boldsymbol{V} \rangle$, where $\boldsymbol{U} \in \mathbb{N}^{M \times 2}$ are the indices of nonzero values, and $\boldsymbol{V} \in \mathbb{R}^{M \times F}$ are the nonzero values. With this representation, for $m = 1, \ldots, M$, we have $\boldsymbol{V}_m = \mathbf{X}_{\boldsymbol{U}_m}$.

Each of the contraction operations of Section 3.2.3 for a tensor $\mathbf{X}^{\mathrm{r}}$ can be implemented with a space complexity of $\mathcal{O}(M + N_{\overline{r}} + N_{\underline{r}})$ and a time complexity of $\mathcal{O}\left(M + N_{\overline{r}} + N_{\underline{r}}\right)$.

The sparse versions of the expansion operations of Section 3.2.3 only output to the desired indices of the sparse output tensor, $\mathrm{sp}(\mathbf{Y}^{\mathrm{r}'}) = \langle \boldsymbol{U}', \boldsymbol{V}' \rangle$, $\boldsymbol{U}' \in \mathbb{N}^{M' \times 2}$, $\boldsymbol{V}' \in \mathbb{R}^{M' \times F}$. So, for example, the sparse "broadcast over rows" operation $\boldsymbol{Y}^{r'} = \mathbf{1}_{N_{\overline{r}'}} \boldsymbol{z}^{\overline{r}\top}$ would instead be implemented so that for $m' = 1, \ldots, M'$, if $(i, j) = \boldsymbol{U}'_{m'}$ then $\boldsymbol{V}'_{m'} = \boldsymbol{z}^{\overline{r}}_j$. The sparse "identity" operation $\boldsymbol{Y}^{\langle \overline{r}', \underline{r}' \rangle} = \boldsymbol{Z}^{\langle \overline{r}, \underline{r} \rangle}$ would instead be implemented as $\boldsymbol{V}'_{m'} = \boldsymbol{V}_m$ if $\exists m, \boldsymbol{U}'_{m'} = \boldsymbol{U}_m$, else 0. The sparse expansion operations have a space complexity of $\mathcal{O}(M + M' + N_{\overline{r}} + N_{\underline{r}} + N_{\overline{r}'} + N_{\underline{r}'})$ and a time complexity of $\mathcal{O}\left((M + M') \log(M + M') + N_{\overline{r}} + N_{\underline{r}} + N_{\overline{r}'} + N_{\underline{r}'}\right)$. The log-factor is because the "identity" and "transpose" expansion operations require that we match the nonzero indices of the input and output matrices, an operation that involves sorting, giving it a time complexity of $\mathcal{O}((M + M') \log(M + M'))$. However, this only needs to be computed once for a given input and output sparsity mask, rather than for every pass through a layer.

With a sparse implementation, the layer effectively has a linear complexity in the number of nodes and edges of the graph, making it efficient for large datasets. However, inducing the sparsity on the output of the layer should be seen as an non-linear operation. Because permutation of node types also permutes the sparsity patterns, this non-linear operation is equivariant.

3.3.3 Encoding and Decoding Layers

It is also useful to have matrix-to-vector encoding and vector-to-matrix decoding layers. For example, an encoding layer can take in a set of adjacency matrices and output embeddings for each node of each type, while a decoding layer can take in node embeddings and output values for each possible edge, which may be used for link prediction.

We define here an equivariant encoding mapping $\mathrm{P}^{\mathrm{r}\to d} : \mathbb{R}^{N_{\overline{r}}\times N_{\underline{r}}} \to \mathbb{R}^{N_d}$ and an equivariant decoding mapping $\mathrm{B}^{d\to\mathrm{r}} : \mathbb{R}^{N_d} \to \mathbb{R}^{N_{\overline{r}}\times N_{\underline{r}}}$. The equivariance conditions on these mappings are:

$$\mathrm{P}^{\mathrm{r}\to d}(\pi_{\overline{r}}\boldsymbol{X}^{\mathrm{r}}\pi_{\underline{r}}^{\top}) = \pi_d \mathrm{P}^{\mathrm{r}\to d}(\boldsymbol{X}^{\mathrm{r}}) \quad \forall \pi_{\overline{r}}, \pi_{\underline{r}}, \pi_d \in S(N_{\overline{r}}) \times S(N_{\underline{r}}) \times S(N_d) \tag{3.5}$$

$$\mathrm{B}^{d\to\mathrm{r}}(\pi_d \boldsymbol{z}^d) = \pi_{\overline{r}} \mathrm{B}^{d\to\mathrm{r}}(\boldsymbol{z}^d)\pi_{\underline{r}}^{\top} \quad \forall \pi_d, \pi_{\overline{r}}, \pi_{\underline{r}} \in S(N_d) \times S(N_{\overline{r}}) \times S(N_{\underline{r}}) \tag{3.6}$$

Theorem 3.3.1. *Given an edge type* $\mathrm{r} = \langle \overline{r}, \underline{r} \rangle$ *and a node type* d*, all the linear maps* $\mathrm{P}^{\mathrm{r}\to d} : \mathbb{R}^{N_{\overline{r}}\times N_{\underline{r}}} \to \mathbb{R}^{N_d}$ *that satisfy the equivariance condition of Eq.* (3.5) *are produced using the valid contractions of the forms* $\mathbb{R}^{N_{\overline{r}}\times N_{\underline{r}}} \to \mathbb{R}^{N_d}$ *in Section 3.2.3, or contractions of the form* $\mathbb{R}^{N_{\overline{r}}\times N_{\underline{r}}} \to \mathbb{R}$ *followed by a multiplication by* $\mathbf{1}_{N_d}$*.*
All the linear maps $\mathrm{B}^{d\to\mathrm{r}} : \mathbb{R}^{N_d} \to \mathbb{R}^{N_{\overline{r}}\times N_{\underline{r}}}$ *that satisfy the equivariance condition of Eq.* (3.6) *are produced using the valid expansions of the form* $\mathbb{R}^{N_d} \to \mathbb{R}^{N_{\overline{r}}\times N_{\underline{r}}}$ *in Section 3.2.3, or by first multiplying by* $\mathbf{1}_{N_d}^{\top}$ *and then applying expansions of the form* $\mathbb{R} \to \mathbb{R}^{N_{\overline{r}}\times N_{\underline{r}}}$*.*

Proof. We can use a similar proof as Theorem 3.2.1, by verifying that the number of operations matches the numbers proved by Maron, Ben-Hamu, Shamir, et al. 2018. For $\mathrm{P}^{\mathrm{r}\to d}$, when $\overline{r} = \underline{r} = d$, then we can count 3 contraction operations of the form $\boldsymbol{X} \to \boldsymbol{z}$ and 2

of the form $\boldsymbol{X} \to z$, yielding the 5 operations predicted by Eq. (5.6). Likewise, for $\mathrm{B}^{d\to \mathrm{r}}$, when $\overline{r} = \underline{r} = d$, we can count 3 expansion operations of the form $\boldsymbol{z} \to \boldsymbol{Y}$ and 2 of the form $z \to \boldsymbol{Y}$, yielding the 5 operations predicted by Eq. (5.6).

Again, we can verify that the correspondence between the number of valid operations and the number predicted by Eq. (5.6) holds for each of the 5 possible sets of equalities between $\overline{r}, \underline{r}$, and d.

As with Theorem 3.2.1, these are also special cases of Theorem 5.1.1.

□

As with the standard equivariant layers we've defined, these equivariant mappings can be combined together to form layers. Here, we define $\mathbb{F} = \{\boldsymbol{z}^d \in \mathbb{R}^{N_d} | d \in \mathbb{D}\}$ to be the set of node feature vectors for each node type.

$$\mathrm{P}^{\mathbb{R}\to\mathbb{D}}(\mathbb{X}) = \left\{ \sum_{\boldsymbol{X}^{\mathrm{r}} \in \mathbb{X}} \sum_{c} \mathrm{P}_c^{\mathrm{r}\to d}(\boldsymbol{X}^{\mathrm{r}}) \mid d \in \mathbb{D} \right\} \tag{3.7}$$

$$\mathrm{B}^{\mathbb{D}\to\mathbb{R}}(\mathbb{F}) = \left\{ \sum_{\boldsymbol{z}^d \in \mathbb{F}} \sum_{c} \mathrm{B}_c^{d\to \mathrm{r}}(\boldsymbol{z}^d) \mid \mathrm{r} \in \mathbb{R} \right\} \tag{3.8}$$

where for P, c indexes each valid contraction operation, and for B, c indexes each valid expansion operation.

3.3.4 Sharing Weights

The model described above learns independent parameters for each pair of edge types. However, in some datasets, the relations may have something semantic meaning in common, and it could be good to share parameters. This may be accomplished by replacing the weight $w_c^{\mathrm{r}\to\mathrm{r}'}$ in Equation 3.4 with $(w_c^{\mathrm{r}\to\mathrm{r}'} + w_c)$, where w_c is an additional weight for operation c shared for all combinations of edge types.

4

Model Evaluation

4.1 Tasks and Architectures

Our heterogeneous graph layers, just like regular linear layers in a multilayer perceptron, can be stacked together and alternated with nonlinear activation functions to form a variety of neural network architectures. We call the resulting neural networks "Equivariant Heterogeneous Neural Networks", or E-HGNNs. We describe here two conventional graph learning tasks, and what architectures we designed for them.

4.1.1 Node Classification

For the task of node classification, we are provided with a heterogeneous graph, where a subset of nodes of one target type are labelled. We are tasked with predicting the labels of the other nodes of the target type.

We use an architecture consisting of a stack of equivariant heterogeneous graph layers separated by nonlinear activation functions. We apply batch normalization over the nonzero entries for each of the matrices at each layer, and we use channel-wise dropout at each layer to prevent overfitting. Following our stack of equivariant heterogeneous graph layers, we add an encoding layer. This encoding layer can either be used to directly predict classes for each node, or they can be used to get an embedding vector for each node which is then fed into a

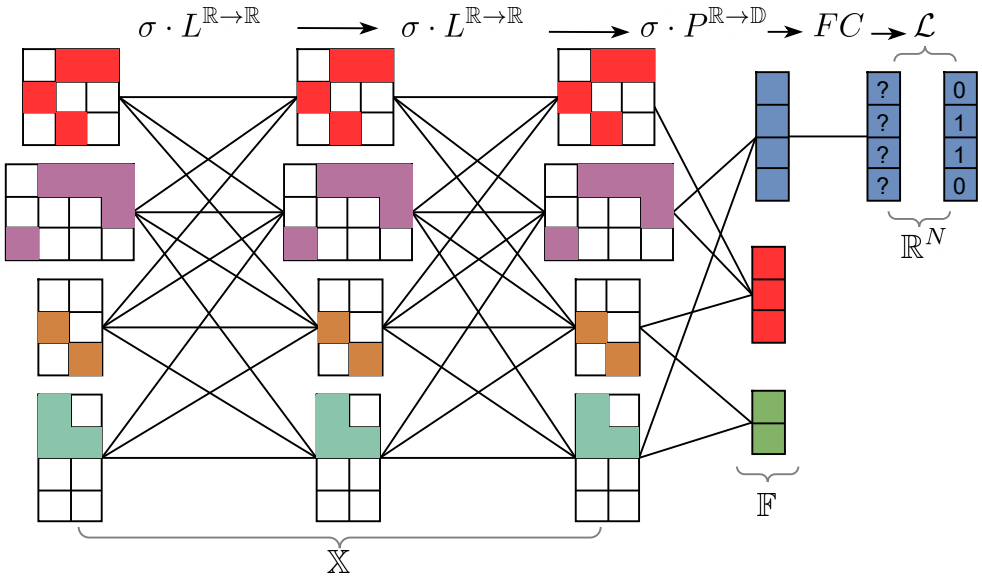


Figure 4.1: Diagram of a two-layer node classification architecture, being trained on the task of classifying the PUBLICATION nodes (●) from the example dataset in Fig. 1.1. FC denotes a fully-connected linear layer, and $\mathcal{L}$ denotes the loss function. Some details are omitted for simplicity: connections between relations without any node types in common are not shown, and node features from the input graph are not shown.

conventional linear classifier to get predictions. The network is trained using a negative log loss over labels.

4.1.2 Link Prediction

For the task of link prediction, we are provided with a heterogeneous graph where a subset of edges of one target type have been removed. Given a set of candidate edges, we are tasked with assigning a confidence score to each potential edge. Half of the candidate edges are real, and half are constructed by taking a real edge and replacing one node with a random 2-hop neighbour of the other node.

To accomplish this task, we use an autoencoder architecture. We create a stack of equivariant heterogeneous graph layers separated by nonlinearities, followed by an encoding layer that produces node embeddings for each node of each node type. This makes up the encoding module of our autoencoder. These node embeddings are then passed into a decoding layer, producing matrices for each edge type. These matrices are passed through

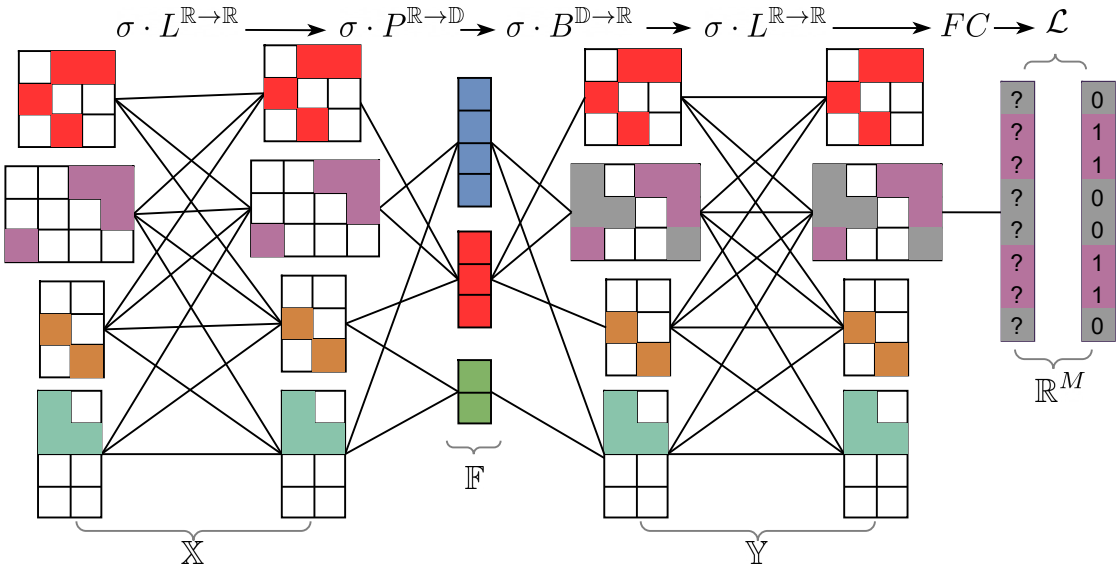


Figure 4.2: Diagram of an autoencoding link prediction architecture with a single layer per module, being trained on the task of predicting AUTHOR-PUBLICATION (◆ - ●) links from the example dataset in Fig. 1.1. Connections between relations without any node types in common are not shown, and node features from the input graph are not shown. The grey squares indicate fake training sample edges.

another stack of equivariant heterogeneous graph layers, outputting a confidence score for each potential edge for the target edge type. The neural network is trained using binary cross-entropy loss over a 1:1 mix of samples of real edges of the target edge type, and randomly sampled fake edges.

4.2 Heterogeneous Graph Benchmark

We evaluate our architectures using the recently created Heterogeneous Graph Benchmark (HGB) (Lv et al. 2021), which gives a set of standardized datasets and training/test splits in node classification and link prediction. To prevent any possible test set leakage, test set labels are withheld, and evaluation metrics are obtained by submitting predictions to the HGB website[1]. We make comparisons against the heterogeneous graph neural networks Simple-HGN (Lv et al. 2021), RGCN (Schlichtkrull et al. 2018), HAN (X. Wang, Ji, et al. 2019),

[1] https://www.biendata.xyz/hgb/

GTN (Yun et al. 2019), RSHN (S. Zhu et al. 2019), HetGNN (Zhang et al. 2019), MAGNN (X. Fu et al. 2020), HetSANN (Hong et al. 2020), HGT (Hu et al. 2020), and the homogeneous graph neural networks GCN (Kipf and Welling 2016), and GAT (Veličković et al. 2017). All evaluation scores listed here are taken from Lv et al. 2021. Details on the specific hyperparameters searched over and used for each dataset are included in Section 4.2.3.

Table 4.1: Characteristics of each of the datasets tested.

Node Classification	Nodes	Node Types	Edges	Edge Types	Node Attributes	Target	Classes
DBLP	26,128	4	239,566	6	Yes	Author	4
IMDB	21,420	4	86,642	6	Yes	Movie	5
ACM	10,942	4	547,872	8	Yes	Paper	3
Freebase	180,098	8	1,057,688	36	No	Book	7
Link Prediction						Target	
Amazon	10,099	1	148,659	2	Yes	Product-product	
LastFM	20,612	3	141,521	3	No	user-artist	

4.2.1 Node Classification

We look at four node classification datasets: DBLP, IMDB, ACM, and Freebase. DBLP, ACM, and Freebase are multi-class classification tasks, and IMDB is a multi-label task. All datasets except for Freebase additionally include node attributes, and 24% of target nodes labels are used for training, 6% for validation, and 70% for testing. Further information is included in Table 4.1, which is adapted directly from Lv et al. 2021. The task is evaluated using the metrics of Micro-F1 and Macro-F1 scores (F1 scores that have been averaged over all nodes and all labels respectively).

A comparison between our results and competing methods is shown in Table 4.2. It can be seen that our method generally performs comparably with other top methods, and yields higher performance than the state of the art for two particular metrics.

Table 4.2: Comparison of our method on the node classification task.

	DBLP		IMDB	
Method	Macro-F1	Micro-F1	Macro-F1	Micro-F1
Simple-HGN	**94.01±0.24**	**94.46±0.22**	62.05±1.36	**67.36±1.36**
RGCN	91.52±0.50	92.07±0.50	58.85 ± 0.26	62.05±0.15
HAN	91.67±0.49	92.05±0.62	57.74±0.96	64.63±0.58
GTN	93.52±0.55	93.97±0.54	60.47±0.98	65.14±0.45
RSHN	93.34±0.58	93.81±0.55	59.85±3.21	64.22±1.03
HetGNN	91.76±0.43	92.33±0.41	48.25±0.67	51.16±0.65
MAGNN	93.28±0.51	93.76±0.45	56.49±3.20	64.67±1.67
HetSANN	78.55±2.42	80.56±1.50	49.47±1.21	57.68±0.44
HGT	93.01±0.23	93.49±0.25	63.00±1.19	67.20±0.57
GCN	90.84±0.32	91.47±0.34	57.88±1.18	64.82±0.64
GAT	93.83±0.27	93.39±0.30	58.94±1.35	64.86±0.43
E-HGNN (ours)	92.79±0.33	93.29±0.3	**63.15±1.06**	66.67 ±0.92
	ACM		**Freebase**	
Method	Macro-F1	Micro-F1	Macro-F1	Micro-F1
Simple-HGN	**93.42±0.44**	**93.35±0.45**	47.72±1.48	**66.29±0.45**
RGCN	91.55±0.74	91.41±0.75	46.78±0.77	58.33±1.57
HAN	90.89±0.43	90.79±0.43	21.31±1.68	54.77±1.40
GTN	91.31±0.70	91.20±0.71	-	-
RSHN	90.50±1.51	90.32±1.54	-	-
HetGNN	85.91±0.25	86.05±0.25	-	-
MAGNN	90.88±0.64	90.77±0.65	-	-
HetSANN	90.02±0.35	89.91±0.37	-	-
HGT	91.12±0.76	91.00±0.76	29.28±2.52	60.51±1.16
GCN	92.17±0.24	92.12±0.23	27.84±3.13	60.23±0.92
GAT	92.26±0.94	92.19±0.93	40.74±2.58	65.26±0.80
E-HGNN (ours)	92.26±0.44	92.17 ±0.45	**48.35±1.57**	63.42±0.29

4.2.2 Link Prediction

We look at two link prediction datasets: Amazon and LastFM. The Amazon dataset additionally include node attributes. For each dataset, 81% of edges of the target edge type are used for training, 9% are used for validation, and 10% are withheld for the test set. The fake edges used for training were sampled randomly, but for validation and testing, the fake edges were sampled from the set of 2-hop neighbours of each node. Further information on these datasets is included in Table 4.1.

Edge predictions are evaluated using two metrics: The area under the Receiver Operating Characteristic curve (ROC-AUC), and the Mean Reciprocal Rank (MRR). The ROC-AUC score evaluates the model's ability to discriminate between real and fake edges over different

sensitivity thresholds. The MRR score evaluates the model's ability to rank real candidate edges higher than false edges.

A comparison between our results and competing methods is shown in Table 4.3. Our method performs comparably to other leading methods on the LastFM benchmark nad outcompetes all other methods on the Amazon benchmark

Table 4.3: Comparison of the Equivariant HGN architecture on the link prediction task.

	Amazon		**LastFM**	
Method	ROC AUC	MRR	ROC AUC	MRR
Simple-HGN	93.40±0.62	96.94±0.29	**67.59±0.23**	**90.81±0.32**
RGCN	86.34±0.28	93.92±0.16	57.21±0.09	77.68±0.17
GATNE	77.39±0.50	92.04±0.36	66.87±0.16	85.93±0.63
HetGNN	77.74±0.24	91.79±0.03	62.09±0.01	83.56±0.14
MAGNN	-	-	56.81±0.05	72.93±0.59
HGT	88.26±2.06	93.87±0.65	54.99±0.28	74.96±1.46
GCN	92.84±0.34	97.05±0.12	59.17±0.31	79.38±0.65
GAT	91.65±0.80	96.58±0.26	58.56±0.66	77.04±2.11
E-HGNN (ours)	**96.75±0.16**	**97.78±0.15**	60.94±0.36	82.36±0.71

The HGB also includes a third dataset, PubMed. Our model is able to perform near-perfectly on this dataset, but this is because of an improper train/test split in the benchmark: the dataset has a fully symmetric adjacency matrix for the test edge type, and both edges in both directions were included in the dataset. The equivariant HGN at test time could therefore determine if a set of test edges are real or not by checking whether their corresponding inverse edges exist in the training set. Due to this data leakage, its performance is not included here.

4.2.3 Hyperparameters

For both the link prediction and the node classification task, we used the Adam Optimizer with weight decay (Kingma and Ba 2014). We also optionally apply a fully connected layer to the graph node attributes before passing it on to the rest of our network. For each dataset, we ran sweeps on a range of hyperparameters, evaluating their performance against a held-out validation set. The hyperparameters used and the range we tested over are included

in Table 4.4. The sets of hyperparameter values that yielded the best performance on the validation set for each dataset are included in Table 4.5.

Table 4.4: Hyperparameters tested for each task.

Task	Hyperparameter	Description	Sweep Range
Both	ACT_FN	Nonlinear activation function	ReLU, LeakyReLU, Tanh
	DROPOUT	Channel-wise dropout	0, 0.1, 0.3, 0.5
	LR	Optimizer learning rate	1e-3, 5e-4, 1e-4
	POOL_OP	Pooling operation used instead of the inner summation in Eq. (3.4)	mean, max
	WEIGHT_DECAY	Optimizer weight decay	1e-3, 1e-4, 1e-5, 1e-6
	WIDTH	Number of feature dimensions for each equivariant layer	16, 32, 64
Node Classification	DEPTH	Number of equivariant layers + optional input fully connected layer	1, 2, 3, 4, 5, 6
	FC_LAYER	Input dimension of optional additional fully connected layer after obtaining node embeddings	0, 16, 32, 64, 128
	FEATS_TYPE	If True, ignore node features of non-target nodes	True, False
	IN_FC_LAYER	If True, the first layer of the network is set to be a fully connected layer instead of an equivariant layer	True, False
Link Prediction	DEPTH	Number of equivariant and fully connected layers in both the encoding and decoding modules	2, 3, 4, 5, 6
	EMBEDDING_DIM	Dimensions of node embeddings	32, 64, 128
	IN_FC_LAYER	If True, the first layer of the encoding module and the last layer of the decoding module are set to be fully connected layers instead of equivariant layers	True, False

Table 4.5: Hyperparameters selected for each dataset for both tasks.

	Node Classification				Link Prediction	
	DBLP	**IMDB**	**ACM**	**Freebase**	**Amazon**	**LastFM**
ACT_FN	LeakyReLU	LeakyReLU	ReLU	Tanh	ReLU	ReLU
DEPTH	6	6	3	6	5	4
DROPOUT	0	0.3	0.3	0	0.1	0
EMBEDDING_DIM	-	-	-	-	64	64
FC_LAYER	32	128	64	16	-	-
FEATS_TYPE	True	True	False	True	-	-
IN_FC_LAYER	False	False	False	True	True	True
LR	0.001	0.0001	0.001	0.0005	0.001	0.001
POOL_OP	max	mean	mean	mean	mean	mean
WEIGHT_DECAY	0.001	0.0001	1.00E-05	1.00E-06	0.001	0.0001
WIDTH	64	64	64	16	128	128

4.3 Synthetic Datasets

Heterogeneous graph datasets can vary in many different ways, so it can be difficult to assess which graph neural network models are useful under what circumstances. The performance of the E-HGNN model relative to other benchmark methods varied on HGB, and since the datasets in HGB are so different (see 4.1), it is not clear what caused differences in performance. One way to account for these aspects is by applying the model to heterogeneous graph datasets, where each of these features may be varied.

4.3.1 Dataset Generation

The synthetic datasets described in this section are based on the Multiple Random Dot Product formulation for a graph (S. Wang et al. 2021; C. L. M. Nickel 2008; Nielsen and Witten 2018). In this model, a node v_i of a graph is assumed to possess some underlying vector $\boldsymbol{z}_i \in \mathbb{R}^h$. Each edge type r in a heterogeneous graph is represented by a diagonal relation matrix $\boldsymbol{J}$. To generate an adjacency matrix $\boldsymbol{A}$ for edge type r, we can use a function of the product of the node vectors and the relation matrix, i.e.: $P(\boldsymbol{A}_{i,j} = 1) = f(\boldsymbol{z}_i^\intercal \boldsymbol{J} \boldsymbol{z}_j)$ for some function $f : \mathbb{R} \to [0, 1]$. In order to control whether a relation is homophilic or heterophilic, $\boldsymbol{J}$ may be broken down into $\boldsymbol{J} = \boldsymbol{J}_+ \boldsymbol{K}$ where $\boldsymbol{J}_+$ is a diagonal matrix with only positive entries, and where $\boldsymbol{K}$ is a diagonal matrix whose entries are either -1

with probability p_{het} or 1 with probability $1 - p_{\text{het}}$. When $p_{\text{het}} = 0$, then $\boldsymbol{z}_i^\intercal \boldsymbol{J} \boldsymbol{z}_j$ can be interpreted as the similarity between i and j, with each dimension of their underlying vectors weighted by an entry in $\boldsymbol{J}$. When $p_{\text{het}} = 1$, then this product can instead be interpreted as the dissimilarity between i and j, yielding heterophilic graphs. By allowing p_{het} to vary continuously, the resulting synthetic graphs can be somewhat smoothly varied in how hetero- or homophilic they are. After obtaining the scores $\boldsymbol{M}_{i,j} = \boldsymbol{z}_i^\intercal \boldsymbol{J} \boldsymbol{z}_j$, three different functions $f(\boldsymbol{M})$ were tested for transforming them into adjacency matrices for a given density (ratio of edges M to possible edges $N \times N$):

- Proportional: the sigmoid function is used to derive an edge probability for each index, which is then normalized by the graph sparsity: $f(\boldsymbol{M}) = \sigma(\boldsymbol{M}) \times \text{density}$.
- Uniform: All indices where $\boldsymbol{M}_{i,j} > 0$ are sampled with a probability of $M/|\{\boldsymbol{M}_{i,j} > 0\}|$.
- Threshold: Given a desired graph sparsity, a percentile of $(1 - \text{sparsity})$ is used as a hard threshold, with all scores above the threshold yielding an edge.

4.3.2 Architectures and Dataset Parameters

Using the datasets above, a set of experiments were created to test out the E-HGNN model under different dataset conditions. The E-HGNN model was compared to the baseline models of GCN (Kipf and Welling 2016), and GAT (Veličković et al. 2017). For GCN and GAT, node and edge type information is entirely ignored, essentially treating the input graph as homogeneous. These relatively simple baselines are used instead of any neural networks specifically designed for heterogeneneous graphs because they have been shown to perform competitively in this situation by Lv et al. 2021, whose results are shown in Table 4.3.

Unless specified otherwise, each synthetic graph used for experiments has 1000 nodes per node type, a density of 0.01, two node types, four edge types (corresponding to relations linking each node type to each other), and is entirely homophilic (i.e. $p_{\text{het}} = 0$). The following four experiments were conducted, each with the goal of observing the effects of varying a single graph parameter:

- The number of node types $D = |\mathbb{D}|$ is varied between 1, 2, and 3. A relation is created from every node type with every other node type, i.e. $\mathbb{R} = \{\langle i, j\rangle, \forall i, j \in \mathbb{D}\}$.
- The number of edge types $R = |\mathbb{R}|$ is varied between 1, 4, and 9. Only one node type is used, so each relation is from that node type back to itself.
- The graph density is varied. In order to keep the total number of edges constant, the density changes with the square of the number of nodes per node type. Densities of 0.000625, 0.0025, 0.01, 0.04, and 0.16 were used, with 4000, 2000, 1000, 500, and 250 nodes per node type respectively.
- Level of heterophily, controlled by p_{het}, is varied. Values of 0, 0.5, and 1 were tested.

Each experiment was ran with 5 different random seeds for each model type. The hyperparameters used for each model were not tuned for the tasks and were instead just set to their defaults, as these experiments are not to compare the real performance of the models against each other, but rather to compare how their performances change under changing datasets.

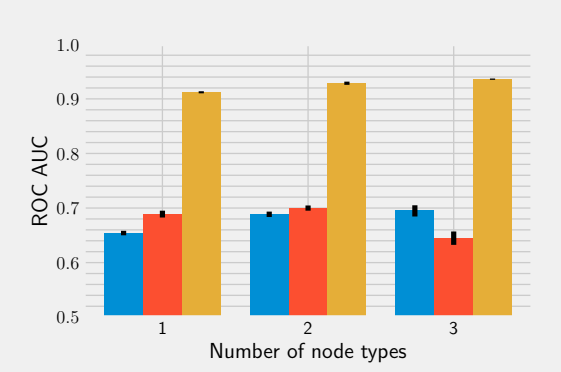
4.3.3 Link Prediction

We tested the models on these synthetic datasets on the task of link prediction, where we are predicting whether edges exist within a single edge type (the target edge type). The target edge type is always for a relation from one node type to itself. Nodes were constructed using 40-dimensional hidden vectors, and edges were created using the threshold method described in Section 4.3.1. Additional 2-dimensional node attributes were created by multiplying each node's hidden vector by a randomly generated 40 by 2 matrix, and these node attributes were used as inputs to the neural networks.

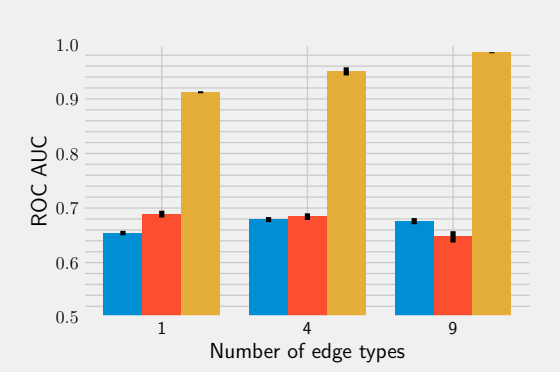
All models used a weight decay of 1e-4, used mean pooling, used LeakyReLU as an activation function, and had a depth of 3 layers and a width of 64 channels. The E-HGNN network was created using an autoencoding architecture with a 64-dimensional node embeddings to directly predict edges. The GCN and GAT models were used to produce 64-dimensional

node embeddings, and the dot products of the embeddings were used as logits for predicting whether those two nodes have an edge between them.

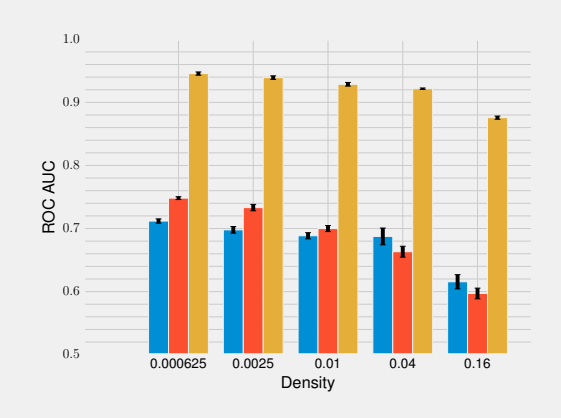
All models were trained for 300 epochs with a learning rate of 0.001, and they were evaluated on a mix of a set of withheld edges equal to 20% of the original graph edges, and a set of edges between nodes and randomly selected 2-hop neighbours. The results of the four experiments are shown in Fig. 4.3, using the ROC-AUC metric.



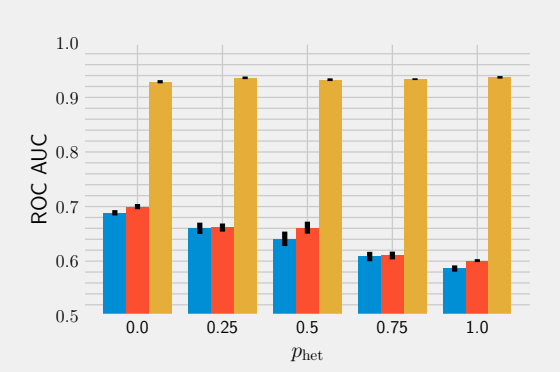
(a) Varying number of node types. The number of edge types is the square of the number of entities.



(b) Varying the number of edge types, while keeping the number of node types fixed at 1.



(c) Varying the density of edges for each edge type. There are 2 node types and 4 edge types.



(d) Varying the level of heterophily of each edge type's relation. There are 2 node types and 4 edge types

Figure 4.3: Results of link prediction experiments. GCN is shown in blue (■), GAT is shown in red (■), and E-HGNN is shown in yellow (■). Standard error is shown.

4.3.4 Results

From Fig. 4.3a and Fig. 4.3b, it is clear that E-HGNN benefits from the additional information provided by having more node types and more edge types. It is able to acquire additional information about the the nodes involved in the target edge type's relation by using multiple edge types (in Fig. 4.3b), and to a lesser extent it can learn node information from edges with other node types (in Fig. 4.3a). With the baseline models, it appears that adding more entities or edge types has little effect on the test scores.

From Fig. 4.3d, it appears that E-HGNN performs about equally well regardless of whether an edge type's relation is homophilic, heterophilic, or in between. In contrast, the performance of both GCN and GAT deteriorate when relations are less homophilic. The failure of conventional GNNs to generalize to heterophilic graphs has been noted by (J. Zhu, Y. Yan, et al. 2020; J. Zhu, Rossi, et al. 2021; Zheng et al. 2022).

Lastly, from Fig. 4.3c, it appears that all models actually perform better on larger, sparser graphs than they do on smaller dense ones. This result is counterintuitive, as we would expect networks to more easily pick up each node's information when each node has more edges. However, this difference might be specific to the specific dataset generation procedure used: using the threshold graph generation technique, sparser graphs will only contain links between very similar nodes, while denser graphs will contain many non-informative links. Thus, the link prediction task is easier for sparser graphs.

5

Heterogeneous Hypergraphs

In this chapter, the theory of Chapter 3 is extended to hypergraphs, where a *hyperedge type* $\mathrm{r} = \langle r(1), \ldots, r(k) \rangle$ relates k node types. What was previously a set of matrices is now a set of tensors $\mathbb{X} = \{\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N_{r(1)} \times \ldots \times N_{r(k)}} \mid \mathrm{r} \in \mathbb{R}\}$. Note that k should really be written as k_{r} since it can be different for each relation, but we omit the subscript since it is clear from context.

5.1 Equivariant Maps for Heterogeneous Hypergraphs

Similar to Eq. (3.4) in heterogeneous graphs, the linear map for the heterogeneous hypergraph setup also decomposes into blocks, and it is sufficient to identify the form of equivariant linear maps $\mathrm{L}^{\mathrm{r}\to\mathrm{r}'} : \mathbf{X}^{\mathrm{r}} \mapsto \mathbf{Y}^{\mathrm{r}'}$. In this more general case, we wish for $\mathrm{L}^{\mathrm{r}\to\mathrm{r}'}$ to be equivariant to permutations across each of its node type dimensions:

$$\begin{aligned} \mathrm{L}^{\mathrm{r}\to\mathrm{r}'}(\pi_{\mathrm{r}} \cdot \mathbf{X}^{\mathrm{r}}) &= \pi_{\mathrm{r}'} \cdot \mathrm{L}^{\mathrm{r}\to\mathrm{r}'}(\mathbf{X}^{\mathrm{r}}), \\ \forall \pi_{\mathrm{r}} \in S(N_{\mathrm{r}(1)}) \times \cdots \times S(N_{\mathrm{r}(k)}), \\ \forall \pi_{\mathrm{r}'} \in S(N_{\mathrm{r}'(1)}) \times \cdots \times S(N_{\mathrm{r}'(k')}) \end{aligned} \tag{5.1}$$

Here, π_{r} is no longer a permutation matrix, but instead an operator that permutes the dimensions of each of the node types specified in r.

One way to characterize all such equivariant linear maps is to consider all combinations of pool and broadcast operations with extraction and placement of hyper-diagonals. To facilitate this, these four operations can be described using the following notation:

Pooling $\text{pool}_{\mathbb{P}}(\mathbf{X})$ pools over all the node indices $\mathbb{P}$ of its input – *e.g.*, $\text{pool}_{\{1\}}\boldsymbol{X}^{\langle 2,1\rangle} = \mathbf{1}_{N_2}^\top \boldsymbol{X}^{\langle 2,1\rangle}$ and $\text{pool}_{\{1,2\}}\boldsymbol{X}^{\langle 2,1\rangle} = \mathbf{1}_{N_2}^\top \boldsymbol{X}^{\langle 2,1\rangle}\mathbf{1}_{N_1}$.

Broadcasting $\text{broadcast}_{\mathrm{r}'}(\mathbf{Z}^{\mathrm{r}})$ broadcasts the input tensor $\mathbf{Z}^{\mathrm{r}}$ into a tensor corresponding to hyperedge type r'. For example $\text{broadcast}_{\langle 2,1\rangle}\boldsymbol{x}^{\langle 2\rangle} = \boldsymbol{x}\mathbf{1}_{N_1}^\top$. This requires the elements of r to appear in r' – more accurately $\{\!\{\mathrm{r}\}\!\} \subseteq \{\!\{\mathrm{r}'\}\!\}$, where $\{\!\{\mathrm{x}\}\!\}$ denotes a multiset of the elements of tuple x.

Extracting a hyper-diagonal A hyper-diagonal generalizes the notion of diagonal of a matrix. Given a tensor $\mathbf{X}^{\langle r(1),\ldots,r(k)\rangle}$, a hyper-diagonal is identified by a partitioning $\mathbb{H}$ of the set $\{1,\ldots,k\}$ where $r(i) = r(j)$ whenever i, j are in the same partition. With this definition of hyperdiagonal, the extraction operation $\text{extract-diag}_{\mathbb{H}}(\mathbf{X}^{\mathrm{r}})$ reproduces the effect of diag operation for a matrix by simply using $\mathbb{H} = \{\{1,2\}\}$. In the extreme case where $\mathbb{H} = \{\{1\},\ldots,\{k\}\}$ identifies the entire input tensor as the hyper-diagonal, this operation becomes the identity operation.

Placing a hyper-diagonal $\text{place-diag}_{\mathrm{r}',\mathbb{K}}(\mathbf{Z}^{\mathrm{r}})$ places the tensor $\mathbf{Z}^{\mathrm{r}}$ over the hyper-diagonal of a tensor $\mathbf{Y}^{\mathrm{r}'}$ identified by the partition $\mathbb{K}$.

Any equivariant linear operation: $\mathrm{L}^{\mathrm{r}\to\mathrm{r}'} : \mathbb{R}^{N_{\mathrm{r}(1)}\times\ldots\times N_{\mathrm{r}(m)}} \to \mathbb{R}^{N_{\mathrm{r}'(1)}\times\ldots\times N_{\mathrm{r}'(m')}}$ can be written as a linear combination of different "compatible" choices for these four operations:

$$\begin{aligned}\mathbf{Y}^{\mathrm{r}'} &= \mathrm{L}^{\mathrm{r}\to\mathrm{r}'}(\mathbf{X}^{\mathrm{r}}) \\ &= \sum w_{\mathbb{H},\mathbb{P},\mathrm{r}'',\mathbb{K}}\ \text{place-diag}_{\mathrm{r}',\mathbb{K}}\left(\text{broadcast}_{\mathrm{r}''}\left(\text{pool}_{\mathbb{P}}\left(\text{extract-diag}_{\mathbb{H}}\left(\mathbf{X}^{\mathrm{r}}\right)\right)\right)\right)\end{aligned} \tag{5.2}$$

This composition of operations is first extracting a hyper-diagonal using $\mathbb{H}$, pooling over a subset of indices identified by $\mathbb{P}$, broadcasting some of these dimensions and permuting them according to r'', and finally placing the resulting tensor over a hyper-diagonal of the output tensor, identified by $\mathbb{K}$.

Example 2: We give an example of a possible combination of operations when mapping between a relation $\mathrm{r} = \langle 1, 1, 1, 2\rangle$ and $\mathrm{r}' = \langle 1, 3, 3\rangle$.

- If we set $\mathbb{H} = \{\{1, 3\}, \{2\}, \{4\}\}$ then extract-diag$_{\mathbb{H}}(\mathbf{X}^{\langle 1,1,1,2\rangle})$ would return a tensor $\mathbf{Z}^{\langle 1,1,2\rangle} \in \mathbb{R}^{N_1 \times N_1 \times N_2}$ where $\mathbf{Z}^{\langle 1,1,2\rangle}_{i,j,k} = \mathbf{X}^{\langle 1,1,1,2\rangle}_{i,j,i,k}$.
- If we next set $\mathbb{P} = \{1, 3\}$, then pool$_{\mathbb{P}}(\mathbf{Z}^{\{1,1,2\}})$ would return a tensor $\mathbf{Z}^{\{1\}} \in \mathbb{R}^{N_1}$, where $\mathbf{Z}^{\{1\}}_i = \sum_j \sum_k \mathbf{Z}^{\{1,1,2\}}_{j,i,k}$.
- If we next set $\mathrm{r}'' = \langle 1, 3\rangle$, then broadcast$_{\mathrm{r}''}(\mathbf{Z}^{\{1\}})$ would return a tensor $\mathbf{Z}^{\{1,3\}} \in \mathbb{R}^{N_1 \times N_3}$, where $\mathbf{Z}^{\{1,3\}}_{i,j} = \mathbf{Z}^{\{1\}}_i$.
- Finally, if we then set $\mathbb{K} = \{\{1\}, \{2, 3\}\}$, place-diag$_{\mathrm{r}',\mathbb{K}}(\mathbf{Z}^{\{1,3\}})$ returns our output $\mathbf{Y}^{\{1,3,3\}} \in \mathbb{R}^{N_1 \times N_3 \times N_3}$, where $\mathbf{Y}^{\{1,3,3\}}_{i,j,j} = \mathbf{Z}^{\{1,3\}}_{i,j}$.

We could have chosen many other options for $\mathbb{H}, \mathbb{P}, \mathrm{r}''$, and $\mathbb{K}$: this was just one of the $\mathrm{Bell}(3 + 1) \times \mathrm{Bell}(1) \times \mathrm{Bell}(2) = 30$ valid operations for this pair of relations.

The following theorem states that these are the only operations needed to create an equivariant linear layer for homogeneous hyper-graphs.

Theorem 5.1.1. *All equivariant linear maps* $\mathrm{L}^{\mathrm{r}\to\mathrm{r}'} : \mathbb{R}^{N_{\mathrm{r}(1)} \times \ldots \times N_{\mathrm{r}(k)}} \to \mathbb{R}^{N_{\mathrm{r}'(1)} \times \ldots \times N_{\mathrm{r}'(k')}}$ *between two hyperedge types in a hypergraph are of the form Eq.* (5.2).

Proof. It is easy to see that all the four operations used in Eq. (5.2) are equivariant. In order to show that these operations exhaust all possibilities, we count the number of compatible choices for $\mathbb{H}, \mathbb{P}, \mathrm{r}'', \mathbb{K}$ for a given input/output pair of edge types r and r′. Through this counting, we arrive at the same number of operations as what is given by Maron, Ben-Hamu, Shamir, et al. 2018's Theorem 3, where the maximality is also established. A related problem for equivariant linear maps for incidence networks appears in Albooyeh, Bertolini, and Ravanbakhsh 2019, and the following proof is inspired by the combinatorial counting arguments in that paper.

Let $\left\{ {p \atop q} \right\}$ be the number of ways we can partition a set of size p into q non-empty partitions. This is also known as the *Stirling partition number*. Moreover, we use $\kappa(\mathrm{r}, d)$ for $d \in \mathbb{D}$ to

denote the number of occurrences of node type d in edge type r. Now we claim that the total number of compatible choices for $\mathbb{H}, \mathbb{P}, \mathrm{r}'', \mathbb{K}$ is given by

$$\prod_{d=1}^{D} \sum_{k,k'=1}^{\min\{\kappa(\mathrm{r},d),\kappa(\mathrm{r}',d)\}} \left\{ \begin{matrix} \kappa(\mathrm{r},d) \\ k \end{matrix} \right\} \left\{ \begin{matrix} \kappa(\mathrm{r}',d) \\ k' \end{matrix} \right\} \sum_{l=0}^{\min\{k,k'\}} \binom{k}{l} \binom{k'}{l} l! \tag{5.3}$$

Because our operations for each node type are independent, the first product is over all possible node types. In the next summation, we only consider the occurrences of node type d, and partition these in both r and r′ into k and k' non-empty partitions respectively. The subsequent Stirling numbers count the number of ways in which we can produce these partitions. In the inner summation, we select l of these k and k' partitions to match them against each other. The number of such possible choices is given by the number of ways we can select l out of k and k' partitions (given by the Binomial coefficients), times all the possible pairings over these l partitions for the matching purpose ($l!$).

Now that we know what the expression above is counting, let us explain the connection to Eq. (5.2). The intuitive motivation is that we want to enumerate all possible pairings of outputs of extract-diag with inputs of place-diag. In Eq. (5.4), the number k represents the order of the output tensor of extract-diag with node type d, and k' represents the order of the input tensor to place-diag. The Stirling numbers are counting the number of different hyper-diagonals of the input and output tensors $\mathbf{X}^{\mathrm{r}}$ and $\mathbf{Y}^{\mathrm{r}'}$ respectively. Once we identify l of these partitions on hyper-diagonals to match, the remaining dimensions from the input hyper-diagonal are pooled, while we broadcast over those of the output hyper-diagonal.

Now we write the combinatorial expression of Eq. (5.2) in an alternate form:

$$\prod_{d=1}^{D} \sum_{k,k'=1}^{\min\{\kappa(\mathrm{r},d),\kappa(\mathrm{r}',d)\}} \left\{ \begin{matrix} \kappa(\mathrm{r},d) \\ k \end{matrix} \right\} \left\{ \begin{matrix} \kappa(\mathrm{r}',d) \\ k' \end{matrix} \right\} \sum_{l=0}^{\min\{k,k'\}} \binom{k}{l} \binom{k'}{l} l! \tag{5.4}$$

$$= \prod_{d=1}^{D} \sum_{l=0}^{\min\{\kappa(\mathrm{r},d),\kappa(\mathrm{r}',d)\}} \sum_{k=l}^{\kappa(\mathrm{r},d)} \sum_{k'=l}^{\kappa(\mathrm{r},d)} \left(\binom{k}{l} \left\{ \begin{matrix} \kappa(\mathrm{r},d) \\ k \end{matrix} \right\} \right) \left(\binom{k'}{l} \left\{ \begin{matrix} \kappa(\mathrm{r}',d) \\ k' \end{matrix} \right\} \right) l! \tag{5.5}$$

$$= \prod_{d=1}^{D} \mathrm{Bell}(\kappa(\mathrm{r},d) + \kappa(\mathrm{r}',d)) \tag{5.6}$$

where in Eq. (5.5), we simply re-arrange the summations in Eq. (5.4). In arriving at Eq. (5.6) from Eq. (5.5) we use a combinatorial argument: recall that the Bell number $\mathrm{Bell}(k)$ is the

number of different ways we can partition k objects into non-empty partitions. To see why Eq. (5.5) is counting the same number of partitions of $\kappa(\mathrm{r}, d)+\kappa(\mathrm{r}', d)$ objects, first partition each of these two sets into any number $k, k' \geq l$ partitions. Next, merge l of those partitions from the first and second set in all possible ways to create a partitioning of $\kappa(\mathrm{r}, d)+\kappa(\mathrm{r}', d)$ into $k+k'-l$ partitions. It is easy to see that this procedure does not produce the same partitioning twice and all different partitions of $\kappa(\mathrm{r}, d)+\kappa(\mathrm{r}', d)$ are produced in this way. This last expression Eq. (5.6) is what appears in Maron, Ben-Hamu, Shamir, et al. 2018 in Theorem 3. The argument above shows that the number of different ways we can perform Eq. (5.2) is equal to this Bell number and therefore all equivariant linear maps of interest have this form. □

5.2 Higher Order Graph Networks

Working directly with data tensors of higher-order relations would be prohibitively expensive for large graphs with higher order relations. Just as in Section 3.3.2, we can implement tensors $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N_{r(1)} \times \cdots \times N_{r(k)}}$ as the sparse $\mathrm{sp}(\mathbf{X}^{\mathrm{r}}) = \langle \boldsymbol{U}, \boldsymbol{V} \rangle$, with $\boldsymbol{U} \in \mathbb{N}^{M \times k}$ and $\boldsymbol{V} \in \mathbb{R}^{M \times F}$. By combining all valid combinations of the four tensor operations defined in the previous section and operating on sparse tensors, equivariant neural network layers may be created linking relations of any order and involving any node types to one another. Although these networks have not been tested, we believe this to be the first practical formulation of maximal permutation-equivariant linear layers that work with any order and any combination of node types. In this section, we present an informal discussion of possible applications of these networks.

5.2.1 Multi-Order Graph Neural Networks

While this chapter provides a method of creating neural networks for heterogeneous hypergraphs, a number of interesting use cases fall out when we apply these layers to regular homogeneous graphs, or even sets. We can replace the idea of a relation r involving

different node types and just consider the order k of a relation, and consider linear layers $L^{k\to k'} : \mathbb{R}^{N^k\times F} \to \mathbb{R}^{N^{k'}\times F'}$. Using Eq. (5.6), derived by Maron, Ben-Hamu, Shamir, et al. 2018, the number of parameters and separate operations in a $L^{k\to k'}$ layer would be $F \times F' \times \text{Bell}(k + k')$. To put that into perspective, the first 8 Bell numbers are: 1, 1, 2, 5, 15, 52, 203, 877, and 4140, so a single $L^{4\to 4}$ layer using 64 feature channels would require $64 \times 64 \times 4140 = 16,957,440$ parameters.

It was proven by Maron, Fetaya, et al. 2019 that in the most general case, a GNN of the sort described here would need to be $\mathcal{O}(N)$-order in order to act as a universal approximator for a graph of N nodes. While this is obviously infeasible, it is still true that higher values of k yield more expressive models: Maron, Ben-Hamu, Serviansky, et al. 2019 showed that k-order GNNs have performance upper bounded by the k-WL test.

Using this as motivation, we can "lift" a $k = 2$ graph to a higher order $k' > 2$ by using a $L^{k\to k'}$ layer, operate on it with $L^{k'\to k'}$ layers of a network, and then pool down with a $L^{k'\to k}$ layer. This would be similar to the k-WL networks introduced by Morris, Ritzert, et al. 2019, but using maximal linear layers. The downside of this is that the "lifting" operation would result in tensors that are no longer sparse, making it very hard to scale to larger graphs. In order to continue along this research direction, it would be helpful to look at whether higher-order tensors can be made to be sparse, by only looking at local neighbourhoods, and whether doing so would preserve the higher expressivity of higher-order graph networks.

5.2.2 Relational Databases

By being able to consider arbitrary node types in each interaction, heterogeneous hypergraph networks may be applied to a general class of data used in many real-world datasets: the relational database.

Relational databases can be modeled using the entity-relationship model (Chen 1976), where each object belongs to a certain entity types, and the relationships it can take part in are determined by that entity type. Because relationships can involve more than two entity types, this can be considered equivalent to the heterogeneous hypergraph model.

Permutation equivariant neural networks for this data model were first described by Graham, J. Wang, and Ravanbakhsh 2019, where they generate a maximally expressive parameter sharing scheme for linear neural network layers. By implementing this approach using a sparse pooling and broadcasting implementation, our heterogeneous hypergraph networks could be applied to a broad array of real-world datasets, capturing information that was previously unavailable to end-to-end deep learning methods.

6

Discussion

6.1 Applications of E-HGNN

In this thesis, we have presented an efficient implementation of a maximally expressive linear mapping for heterogeneous graphs, and constructed neural networks using these mappings as intermediate layers. When compared to a number of other heterogeneous graph learning algorithms on a standardized baseline, we have seen that the E-HGNN is able to outcompete all other methods on one dataset, and remains competitive on another in its link prediction performance. We further see that it is able to achieve results comparable to state of the art across node classification datasets. No experiments were conducted to test the ability of E-HGNN to perform graph classification on heterogeneous graphs. To our knowledge, there are currently no widely used benchmarks for graph classification on specifically heterogeneous graphs. We are optimistic about the possible performance of our model on this tasks: unlike node classification or link prediction, graph classification is particularly sensitive to both local and global structures of graphs, and because E-HGNN is maximally connected, it has some advantages over message-passing based GNNs. Further work is needed to judge.

From synthetic experiments, it appears that the E-HGNN benefits from side information gleaned from additional node and relation types, and is largely unaffected by the homophily or heterophily of the relations it makes predictions on, or whether different relations in the same heterogeneous graph differ in their degree of homophily. While many standardized

graph network datasets, such as citation networks, have high homophily, many important tasks involve graphs with high degrees of heterophily. Particularly important examples of this are the cases of molecular graphs and protein structures, where homophily cannot be assumed (J. Zhu, Y. Yan, et al. 2020).

A prominent example of a type of heterogeneous graph that was not investigated in this thesis is the knowledge base. Knowledge bases generally contain many unique entities and a very large number of relation types: for example, the Wikidata5m dataset (X. Wang, Gao, et al. 2021) contains 4,594,485 entities with 822 relation types between them. As our E-HGNN model learns a mapping between every relation to every other relation, this would clearly be infeasible for an E-HGNN model without modifications. Knowledge bases may have additional structure to them: the YAGO knowledge base is structured as an ontology, where entities can belong to a hierarchy of classes, and relations are defined between classes (Pellissier Tanon, Weikum, and Suchanek 2020). Further investigation is required to adapt E-HGNN to these situations.

6.2 Further Research

While this thesis has established interesting potential for equivariant heterogeneous layers, there remains many open questions about their applicability, and how they may fit into the broader ecosystem of graph neural networks.

Preliminary investigations have shown that not all of the equivariant linear operations of an E-HGNN layer actually contribute to the model's performance: for example, the expansion operation "Broadcast over rows and columns" can be omitted without any noticeable change in performance. Paring down the number of parameters could improve the model's efficiency and potentially provide a better inductive bias at the expense of expressive power, but further ablation studies are required to determine the influence of each operation.

So far, just two basic neural networks were constructed using the equivariant layer as a basis, but more elaborate networks could be constructed. Modifications that have already

been explored for conventional multilayer perceptrons, such as skip-connections, recurrence, or attention mechanisms, can be applied to equivariant heterogeneous graph layers.

While this thesis considers permutation equivariance in heterogeneous graphs, in practice many graphs represent systems that involve features that may transform with their own symmetries. For example, a molecular graph may have cartesian coordinates associated with each atom, and it would be desirable for a neural network operating on such a graph to be equivariant to rotations and translations of these coordinates (Han et al. 2022). Future work could combine the heterogeneous network architecture presented in this thesis with components that are equivariant to these features.

6.3 Conclusion

By enumerating all equivariant linear operations that can be applied to the data structure of heterogeneous graphs, we have demonstrated how they may be combined to create effective heterogeneous graph neural networks. We demonstrated their effectiveness in the task of link prediction, their competitiveness for node classification, and give an efficient implementation that can be used for any collection of arbitrary interactions between nodes of different types. Furthermore, we describe how to extend this implementation to higher-order heterogeneous graphs, opening up an even larger class of possible tasks.

Due to their ease of implementation and broad applicability, the layers described in this thesis can serve as an exciting jumping-off point as the bases of sophisticated networks that can be applied to a great range of tasks. Broadly speaking, any task on exchangeable sets of objects where we wish to learn different semantics for different types of objects or relations is a potential target for future research with these neural network layers. Battaglia et al. 2018 posit that the relational inductive bias inherent to graph networks make them a critical component of the route towards increasingly complex artificial intelligence that can reason like humans. We hope that by elaborating upon the theory of heterogeneous graph networks

and by presenting a practical implementation in this thesis, we may open up new avenues in this crucial field.

Bibliography

Albooyeh, Marjan, Daniele Bertolini, and Siamak Ravanbakhsh (2019). "Incidence networks for geometric deep learning". In: *arXiv preprint arXiv:1905.11460*.

Azizian, Waïss and Marc Lelarge (2020). "Expressive power of invariant and equivariant graph neural networks". In: *arXiv preprint arXiv:2006.15646*.

Barabási, Albert-László and Eric Bonabeau (2003). "Scale-free networks". In: *Scientific american* 288.5, pp. 60–69.

Battaglia, Peter W. et al. (Oct. 2018). "Relational inductive biases, deep learning, and graph networks". In: *arXiv:1806.01261 [cs, stat]*. arXiv: 1806.01261.

Bronstein, Michael M et al. (2021). "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges". In: *arXiv preprint arXiv:2104.13478*.

Cai, Jin-Yi, Martin Fürer, and Neil Immerman (1992). "An optimal lower bound on the number of variables for graph identification". In: *Combinatorica* 12.4, pp. 389–410.

Chen, Peter Pin-Shan (1976). "The entity-relationship model—toward a unified view of data". In: *ACM transactions on database systems (TODS)* 1.1, pp. 9–36.

Cohen, Taco S et al. (2018). "Spherical cnns". In: *arXiv preprint arXiv:1801.10130*.

Dong, Yuxiao, Nitesh V Chawla, and Ananthram Swami (2017). "metapath2vec: Scalable representation learning for heterogeneous networks". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 135–144.

Dong, Yuxiao, Ziniu Hu, et al. (2020). "Heterogeneous Network Representation Learning." In: *IJCAI*. Vol. 20, pp. 4861–4867.

Dwivedi, Vijay Prakash and Xavier Bresson (2020). "A generalization of transformer networks to graphs". In: *arXiv preprint arXiv:2012.09699*.

Fu, Tao-yang, Wang-Chien Lee, and Zhen Lei (2017). "Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1797–1806.

Fu, Xinyu et al. (2020). "Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding". In: *Proceedings of The Web Conference 2020*, pp. 2331–2341.

Geerts, Floris and Juan L Reutter (2022). "Expressiveness and approximation properties of graph neural networks". In: *arXiv preprint arXiv:2204.04661*.

Gilmer, Justin et al. (2017). "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR, pp. 1263–1272.

Gori, Marco, Gabriele Monfardini, and Franco Scarselli (2005). "A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. IEEE, pp. 729–734.

Graham, Devon, Junhao Wang, and Siamak Ravanbakhsh (2019). "Equivariant entity-relationship networks". In: *arXiv preprint arXiv:1903.09033*.

Haan, Pim de, Taco Cohen, and Max Welling (2020). "Natural graph networks". In: *arXiv preprint arXiv:2007.08349*.

Han, Jiaqi et al. (2022). "Geometrically equivariant graph neural networks: A survey". In: *arXiv preprint arXiv:2202.07230*.

Hartford, Jason et al. (2018). "Deep models of interactions across sets". In: *International Conference on Machine Learning*. PMLR, pp. 1909–1918.

Hong, Huiting et al. (2020). "An attention-based graph neural network for heterogeneous structural learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04, pp. 4132–4139.

Hu, Ziniu et al. (2020). "Heterogeneous graph transformer". In: *Proceedings of The Web Conference 2020*, pp. 2704–2710.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kipf, Thomas N and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907*.

Kondor, Risi et al. (2018). "Covariant compositional networks for learning graphs". In: *arXiv preprint arXiv:1801.02144*.

Kreuzer, Devin et al. (2021). "Rethinking graph transformers with spectral attention". In: *Advances in Neural Information Processing Systems* 34.

LeCun, Yann, Bernhard Boser, et al. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.

LeCun, Yann, Léon Bottou, et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Linmei, Hu et al. (2019). "Heterogeneous graph attention networks for semi-supervised short text classification". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4821–4830.

Liu, Ziqi et al. (2018). "Heterogeneous graph neural networks for malicious account detection". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 2077–2085.

Lv, Qingsong et al. (2021). "Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1150–1160.

Maron, Haggai, Heli Ben-Hamu, Hadar Serviansky, et al. (2019). "Provably powerful graph networks". In: *arXiv preprint arXiv:1905.11136*.

Maron, Haggai, Heli Ben-Hamu, Nadav Shamir, et al. (2018). "Invariant and equivariant graph networks". In: *arXiv preprint arXiv:1812.09902*.

Maron, Haggai, Ethan Fetaya, et al. (2019). "On the universality of invariant networks". In: *International conference on machine learning*. PMLR, pp. 4363–4371.

Masci, Jonathan et al. (2015). "Geodesic convolutional neural networks on riemannian manifolds". In: *Proceedings of the IEEE international conference on computer vision workshops*, pp. 37–45.

Monti, Federico et al. (2017). "Geometric deep learning on graphs and manifolds using mixture model cnns". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5115–5124.

Morris, Christopher, Yaron Lipman, et al. (Dec. 2021). "Weisfeiler and Leman go Machine Learning: The Story so far". In: *arXiv:2112.09992 [cs, stat]*. arXiv: 2112.09992.

Morris, Christopher, Martin Ritzert, et al. (2019). "Weisfeiler and leman go neural: Higher-order graph neural networks". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01, pp. 4602–4609.

Nickel, Christine Leigh Myers (2008). "Random dot product graphs a model for social networks". PhD thesis. Johns Hopkins University.

Nickel, Maximilian et al. (2015). "A review of relational machine learning for knowledge graphs". In: *Proceedings of the IEEE* 104.1, pp. 11–33.

Nielsen, Agnes Martine and Daniela Witten (Nov. 2018). "The Multiple Random Dot Product Graph Model". In: *arXiv:1811.12172 [stat]*. arXiv: 1811.12172.

Pellissier Tanon, Thomas, Gerhard Weikum, and Fabian Suchanek (2020). "Yago 4: A reasonable knowledge base". In: *European Semantic Web Conference*. Springer, pp. 583–596.

Ravanbakhsh, Siamak, Jeff Schneider, and Barnabas Poczos (2017). "Equivariance through parameter-sharing". In: *International Conference on Machine Learning*. PMLR, pp. 2892–2901.

Scarselli, Franco et al. (2008). "The graph neural network model". In: *IEEE transactions on neural networks* 20.1, pp. 61–80.

Schlichtkrull, Michael et al. (2018). "Modeling relational data with graph convolutional networks". In: *European semantic web conference*. Springer, pp. 593–607.

Shi, Chuan et al. (2017). "A survey of heterogeneous information network analysis". In: *IEEE Transactions on Knowledge and Data Engineering* 29.1, pp. 17–37.

Simonovsky, Martin and Nikos Komodakis (2017). "Dynamic edge-conditioned filters in convolutional neural networks on graphs". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3693–3702.

Sun, Yizhou and Jiawei Han (2012). "Mining heterogeneous information networks: principles and methodologies". In: *Synthesis Lectures on Data Mining and Knowledge Discovery* 3.2, pp. 1–159.

Tang, Jian, Meng Qu, and Qiaozhu Mei (2015). "Pte: Predictive text embedding through large-scale heterogeneous text networks". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1165–1174.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems* 30.

Veličković, Petar et al. (2017). "Graph attention networks". In: *arXiv preprint arXiv:1710.10903*.

Wang, Shangsi et al. (Apr. 2021). "Joint Embedding of Graphs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.4. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1324–1336.

Wang, Xiao, Houye Ji, et al. (2019). "Heterogeneous graph attention network". In: *The World Wide Web Conference*, pp. 2022–2032.

Wang, Xiaozhi, Tianyu Gao, et al. (2021). "KEPLER: A unified model for knowledge embedding and pre-trained language representation". In: *Transactions of the Association for Computational Linguistics* 9, pp. 176–194.

Wang, Zifeng et al. (2021). "Online Disease Diagnosis with Inductive Heterogeneous Graph Convolutional Networks". In: *Proceedings of the Web Conference 2021*, pp. 3349–3358.

Weisfeiler, Boris and Andrei Leman (1968). "The reduction of a graph to canonical form and the algebra which appears therein". In: *NTI, Series* 2.9, pp. 12–16.

Wu, Zhenqin, Bharath Ramsundar, et al. (2018). "MoleculeNet: a benchmark for molecular machine learning". In: *Chemical science* 9.2, pp. 513–530.

Wu, Zonghan, Shirui Pan, et al. (2020). "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1, pp. 4–24.

Xu, Keyulu et al. (2018). "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826*.

Yan, Sijie, Yuanjun Xiong, and Dahua Lin (2018). "Spatial temporal graph convolutional networks for skeleton-based action recognition". In: *Thirty-second AAAI conference on artificial intelligence*.

Yang, Carl et al. (2020). "Heterogeneous network representation learning: A unified framework with survey and benchmark". In: *IEEE Transactions on Knowledge and Data Engineering*.

Yu, Bing, Haoteng Yin, and Zhanxing Zhu (2017). "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting". In: *arXiv preprint arXiv:1709.04875*.

Yun, Seongjun et al. (2019). "Graph transformer networks". In: *Advances in Neural Information Processing Systems* 32, pp. 11983–11993.

Zaheer, Manzil et al. (2017). "Deep Sets". In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., pp. 3391–3401.

Zhang, Chuxu et al. (2019). "Heterogeneous graph neural network". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 793–803.

Zheng, Xin et al. (2022). "Graph neural networks for graphs with heterophily: A survey". In: *arXiv preprint arXiv:2202.07082*.

Zhou, Jie et al. (2020). "Graph neural networks: A review of methods and applications". In: *AI Open* 1, pp. 57–81.

Zhu, Jiong, Ryan A Rossi, et al. (2021). "Graph neural networks with heterophily". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12, pp. 11168–11176.

Zhu, Jiong, Yujun Yan, et al. (2020). "Beyond homophily in graph neural networks: Current limitations and effective designs". In: *Advances in Neural Information Processing Systems* 33, pp. 7793–7804.

Zhu, Shichao et al. (2019). "Relation structure-aware heterogeneous graph neural network". In: *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, pp. 1534–1539.

Zitnik, Marinka and Jure Leskovec (2017). "Predicting multicellular function through multi-layer tissue networks". In: *Bioinformatics* 33.14, pp. i190–i198.

A

Summary of Notation

Table A.1: Summary of notation used throughout the thesis.

	Example	Description
General Notation	N, M, F	Upper case: Scalar constants
	x, y, z, etc.	Lower case: Scalar valued variables
	$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$, etc.	Lower case, bold: Vector valued variables
	$\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$, etc.	Upper case, bold: Matrix valued variables
	$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, etc.	Upper case bold sans-serif: Tensor valued variables
	$\mathbb{D}, \mathbb{X}$, etc.	Double stroke: Sets
	$\bigoplus$	Aggregation function, such as summation
	i, j, k, t	Indices
	σ	A nonlinear activation function
	f, ϕ, ψ	Generic functions
	$x, \boldsymbol{x}, \boldsymbol{X}, \mathbf{X}$	Data inputs
	$z, \boldsymbol{z}, \boldsymbol{Z}, \mathbf{Z}$	Intermediate calculations
	$y, \boldsymbol{y}, \boldsymbol{Y}, \mathbf{Y}$	Data outputs
	$w, \boldsymbol{w}, \boldsymbol{W}, \mathbf{W}$	Parameter weights
Mathematical Objects	$\langle \cdot \rangle$	Tuples
	$\{\!\{\cdot\}\!\}$	Multisets
	$\mathbf{1}$	A vector of all ones
	$\boldsymbol{I}$	An identity matrix
	$S(N)$	The symmetric group of N elements
Graphs	$\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$	Graphs, their nodes and edges
	$\boldsymbol{A}$	An adjacency matrix
	π	A permutation matrix or operator
	N, M, F	The number of nodes, edges, and feature dimensions, respectively
	$v \in \mathbb{V}$	Nodes
	$\mathcal{N}(v)$	Neighbours of a node
Heterogeneous Graphs	$d \in \mathbb{D}$	Node types
	$\mathrm{r} \in \mathbb{R}$	Edge types
	$\mathrm{r} = \langle \overline{r}, \underline{r} \rangle$	A relation between node types $\overline{r}$ and $\underline{r}$
	k	The order of a relation
Additional	$\mathrm{L}, \mathrm{P}, \mathrm{B}$	Linear maps
	$\mathrm{sp}(\mathbf{X}) = \langle \boldsymbol{U}, \boldsymbol{V} \rangle$	Sparse representation of $\mathbf{X}$
	$\kappa(\mathrm{r}, d)$	The number of times d appears in tuple r
	$\mathrm{Bell}(k)$	The k-th Bell number