THESIS

DEPARTMENT OF COMPUTING

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

The Symmetric Group and Graph Neural Networks

Author:
John Yao

First Supervisor:
Jeroen Lamb
Second Supervisor:
Kevin Webster

June 22, 2022

Submitted in partial fulfillment of the requirements for the Masters of Engineering in Joint Mathematics and Computing of Imperial College London

Abstract

In graph representational learning, message passing is the dominant paradigm of deep learning approaches, ever since it was introduced in Gilmer 2017 (1). In this paper, we review the standard approach to message passing as well as another key approach we call the global basis method. Past works have explored the theoretical expressive limits of the message passing schema. However, proposed models that reach this limit often either have restrictive assumptions on the source data, or rely on exponential parameter growth. A model for a message passing graph neural network that achieves the theoretical bounds for expressivity is presented, and furthermore this model is shown to generalise many of the existing graph neural network models including many of the most popular state of the art models.

Acknowledgments

For being wise and pleasant mentors, I would like to thank my supervisors, Jeroen Lamb and Kevin Webster, for their much helpful sources of guidance and advice throughout my project. They have always been kind and courteous, yet direct when necessary. They have passed on key paper-writing insights and project management skills that were instrumental in the creation of this paper. Additionally, the belief that algebra, and discrete math will one day form a key backbone of machine learning has admitedlly found a place in my heart as well.
For key support and insights, I would like to thank Victoria Klein, without whom's original idea on CNNs this project would not have been possible. Her insights during our pleasant chats together additionally have helped me greatly along the way to writing this.
For thoroughly proofreading the first draft of this paper and tearing it to shreds, I would like to thank Jonas Scholz. A budding academic could ask for no better friend.
For giving me opportunities to discover the beauty and addiction of math, I'd also like to thank my parents. Looks like math camp paid off, guys.
And for forever being my emotional pillar of support, I would like to thank the rest of my friends and family along the way who have been there for me when I have had no where else to turn throughout my life. You guys rock.

Contents

Chapter 1

Introduction

Machine learning as a field has seen some drastic and exciting improvements in the last few decades in tasks relating to image recognition, natural language processing, financial prediction, robotics, and more. As a field, it is extremely broad due to the wide scope of problems it attempts to solve, and many of the techniques and theoretical framework it relies on have histories stemming from unrelated fields. Due to this, the current landscape of machine learning is a chaotic one, with many independent approaches to develop sound theoretical framework and effective framework for machine learning in the general sense.

What makes the most successful machine learning architectures today work? Bronstein et al. contend in Geometric Deep learning (2) that it is the inclusion of models that account for invariance in the geometric sense that has lead to the results we see today. Including a survey and analysis of some of the most popular machine learning architectures today (2) attempts to unite machine learning under the umbrella of invariances, drawing parallels to the effect of the Erlangen Programme on the field of geometry in the late 1800s. Specifically, Felix Klein's Erlangen Programme attempted to unite the field of geometry–which at the time resembled the machine learning field today in a lack of its complete theory–into the study of invariants. Invariants are mathematical abstractions of symmetries–a geometric term that implies the equality of objects and constructs before and after some action, such as rotation. Today, we examine an aspect of machine learning on graphs, a structure with its own inherent symmetries, through a similar lens.

Invariance has always been known to play a big role in the success of some popular machine learning architectures, such as in convolutional neural networks (CNNs). In CNNs, translation invariance (symmetry of the data with respect to translation) is approximated through convolution, leading to more efficient models. Translational invariance is the idea that if objects shift around in the image they are still recognised and detected for what they are. This, along with a few other key design upgrades, gives CNNs their prominent position today in ML literature. Similar ideas have been expressed in graph machine learning in the context of Graph Neural Networks (GNNs), where we explore symmetry of permutations of nodes in graphs. The applications of invariant theory and in particular, invariance with respect to the symmetric group S_n, on graph neural networks will be explored in this work.

Graph machine learning, as a subset of geometric machine learning, has been an

exciting and rapidly developing field of machine learning in the last few years. Historically, techniques in analysing graph networks of data have stemmed from interests in widely unrelated fields like computational chemistry, network science, and social analysis. New techniques and theory in deep graph representational learning has exploded however, starting with the Deep Sets paper (3) in 2017 and including many different types of neural networks relying on different approaches.
The most common approach and the defining paradigm for graph neural networks is message passing. In the model, local information at graph nodes are "passed" along edges of the graph, aggregated and fed through deep networks to approximate distributions on the nodes. The aggregator function employed in message passing is the topic of interest in this paper, where we present a more general and flexible approach to aggregation using approximation results from invariant theory and discuss its implications.

Chapter 2

Background

Machine learning is most commonly defined as a branch of artificial intelligence, aimed at using data and algorithms to iteratively imitate the process in which humans learn a specific task. In this way, computers can attempt to replicate human effort in different fields, from relatively simple tasks like planning a route between locations, to more complex tasks like diagnosing hospital patients. Noting how broad of a spectrum of situations we are attempting to improve, the range of models and techniques also vary depending on the data and specific learning task.

2.1 The Problem

To be precise, it is important to state formally what mathematical problems we are aiming to solve. Philosophically speaking, it is hard to narrow down what exactly constitutes learning and how machines can imitate it. Modern machine learning has many popular forms, including symbolic reasoning and deep learning. Symbolic reasoning generally deals with logical semantic systems, and logic engineers build systems of evaluation to efficiently resolve queries related to those symbols. Deep learning, the focus of this work, aims to replicate human learning through imitating the way neurons fire information in the human brain. In the world of deep learning, data is iteratively "trained" over to match an evaluation data set. Statisically, this means that the goal of the training is to build an approximation of the data distribution given independent, identically distributed data points and their values of interest. This is known as regression. There is another common task as well, known as classification, where the goal is to build an optimal decision boundary around the data points – again, using the training result data as reference. Without going too much into philosophical detail about the differences, classification can be considered a discrete learning task while regression usually acts on a continuous result space.
What is this training result data? It varies, depending on the task. For example. Given an image classification task, the result data would be the end tags we desire to attach to each given image. Note that this is not a full description of all of the data of the image, but rather only the topic of interest. If our task was to detect whether there were cats in the photo or not, a "True" or "False" tag, encoded as 1 and 0 in a computer system, would be appropriate. Any number of dogs in the image would

not be relevant to this encoding. In an example of regression, given a set of images each with a single cat and an attached "cuteness" value in the real numbers, the result data would be the latter.
In essence, the problem is to find an approximation to these functions, whether discrete or continuous, whose inputs are the data vectors and minimize the "loss"– the distance between the approximation's output value and the training result data value. In the topic of this paper, we examine graphs, their representations, and define below their training tasks and the problems we try to solve.

2.2 Deep Learning

The neural network is the core of deep learning today. The very concept of deep learning originated from a desire to create an automatic feature extraction capability in difficult learning tasks like image and speech recognition. Around the time of the first papers on multilayer architectures circ. 1970 (4), the conventional method for image recognition was through handcrafted feature extractors, built by skilled engineers. The difficulty to this operation can be surmised through the fact that in many instances, selecting for the correct features in the photos that determine the desired classification change can be very subtle. For example, in classifying images of cats, perhaps we desire to classify between siamese cats and ragdoll cats. In both instances the differences in fur color and body shape are extremely subtle and must be picked up on by the feature extractor; meanwhile, bigger aboslute pixel differences such as cat placement in the photo and pose must be ignored as they have no obvious logical connection with the desired classification task. This is what is known as the selectivity-invariance dilemma (4) – producing representations in a normed vector space that are selective to desired aspects of the image but invariant to other changes. The reason why we want to produce representations in said vector space is to be able to utilise a definition of distance, from which we may be able to draw conclusions about the accuracy of our model.
The way deep learning problems are usually presented is as thus: Given data represented in some feature space, and labels representing our desired characteristic of each data point, how can we build a model that takes in new data and *predicts* a new set of labels for the incoming data? Let us illustrate with an example. Assume that we are interested in the valuations of a new set of properties in an up and coming neighbourhood location in a city. Unfortunately, as the properties are new, we have no market data of people buying any properties, so we are left to make evaluations on less concrete foundations. We have access to (1) information such as square footage, distance from local schools, average crime rate, and other assorted information about each new property. We also have (2) the same for other properties that are not new and have been tested on the market and achieved accurate valuations from past purchases. However as an inexperienced real estate company, we do not know anything about how to put this information together in new formulas to obtain an accurate property value estimation (our goal). Deep learning can solve this problem, by allowing neural networks to determine the most relevant function of information on each property that determine value. Relying on our database of

old properties and their sale values (2), we may take that to be our data set. Each old property has all of the information mentioned before represented in some multidimensional vector, known as a *feature*. The labels are their sale prices. The list of feature vectors will be fed into our neural network, and the resultant will be compared to our label. The difference between the expected (label) and the calculated (target) expected value is called the *loss* in deep learning. This loss is then to be minimised in a process called gradient descent. The network model itself is composed of multiple layers, defined below.

Definition 2.2.1 (Linear Layer)**.** Given n data points in an input vector, $x \in \mathbb{R}^n$, a *linear layer* $\mathcal{L}$ is a linear transformation:

$$\mathcal{L}(x) = Ax + b$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, and $\mathcal{L}(x) \in \mathbb{R}^m$.

Definition 2.2.2 (Activation Function)**.** An *activation function* is a nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$, and an *activation layer* is an activation function applied to every element in the input vector, i.e. given an input vector x,

$$\sigma(x) = (\sigma(x_1), ..., \sigma(x_n))$$

The following result shows the power of even a two-layer neural network:

Theorem 2.2.1 (Pinkus 1999 (5))**.** *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous activation function that is not a polynomial. Let $V = \mathbb{R}^d$ be a real finite dimensional vector space. Then any continuous map $f : V \to \mathbb{R}$ can be approximated by uniform convergence on any compact set in V by maps $\hat{f} : V \to \mathbb{R}$ of the form*

$$\hat{f}(x_1, ..., x_d) = \sum_{n=1}^{N} c_n \sigma(\sum_{s=1}^{d} w_{ns} x_s + h_n)$$

for some parameter N and coefficients c_n, w_{ns}, and h_n. Or alternatively,

$$\hat{f}(x) = \mathcal{L}_c(\sigma(\mathcal{L}(x))) \tag{2.1}$$

where $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}^N$ and $\mathcal{L}_c : \mathbb{R}^N \to \mathbb{R}$.

This theorem states, that any continuous function can be approximated to any given error bound by a shallow neural network $\hat{f}$ consisting of two layers and an activation. Non-linear activation layers must be used in between linear layers, as a scalar non-linear function such as the sigmoid function or the rectified linear unit (ReLU). As an informal justification, the concatenation of two linear layers is simply the composition of two linear operators, which is also a layer. Thus having a multi-layer system with no activations would simply behave as a single linear layer, which is not desirable, as this restricts the class of functions able to be learned.
The utility of layers can be seen in the use of backpropagation in machine learning. As stated before, the main advantage of deep learning is its ability to continually adjust its feature extraction, and determine how to best form an approximation of

the desired function without input from the ML engineer. This turns the problem of machine learning into one of unconstrained optimisation of the loss function, given generally by the difference between the target result data values and the approximated function values from the model. As such, the technique most commonly used is gradient descent.
In gradient descent, the loss is propagated throughout the many layers in the model by essentially the chain rule from multivariable calculus. For each layer, the derivative of the loss function is taken with respect to the parameters of the layer and the input of the layer, resulting in a gradient. This is also where linear layers show utility, as the gradient of a linear layer with respect to the input is simply the weights matrix. This gradient, calculated piecewise for every layer in the model working backwards from the loss, is applied by summing its product with some learning rate α, usually a small decimal, with the layer's weights itself. In the case above, the gradient of the layer $\mathcal{L}$ with respect to the inputs, would be A. In this way, the loss would be "propogated" throughout each individual layer, changing each in the direction of greatest descent to the loss. An equation for backpropagation is provided:

Definition 2.2.3 (Backpropagation, Kelly 1960 (6))**.** *Backpropagation* is the updating of the parameters θ of a layer in a neural network at time, by a learning rate α times the derivative of a given loss function L. θ represents both the weights and the bias of the layer in question.

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial L(X, \theta^t)}{\partial \theta}$$

For more details on gradient descent, and the version in use in most literature "Stochastic Gradient Descent", see Amari 1993 (7).

2.3 Graph Representational Learning

The goal of graph representational learning is to infer information about a graph data form. This could mean a single large graph, such as a social network, or a series of smaller graphs such as molecules. Regardless of the task, the techniques used in graph representational learning can be adapted to suit the task at hand, just like in deep learning applications.

Definition 2.3.1 (Graph)**.** A graph $\mathcal{G}$ is defined as $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ are nodes in the graph and $\mathcal{E}$, its edges, are unordered n-tuples of elements from $\mathcal{V}$. n is the fixed dimension of the relation, and in most cases it will be 2. The results in this work pertain to the case of 2-dimensional graphs.

These nodes, edges, and graphs encode information–for example, each node may have features in the space $\mathbb{R}^n$. As a simple example, consider a graph $(\mathcal{V}, \mathcal{E}) = (\{1, 2, 3, 4\}, \{(1, 2), (2, 3), (2, 4)\})$. This graph would be in the shape of a cross, or a plus-sign, with node 2 at its centre and nodes $1, 3,$ and 4 at the ends. As an example of what a feature space for this graph could be, each node's feature could lie in the space $\mathbb{Z}$ and have the degree of the node (the number of neighbouring nodes with

an edge connecting to it) as its feature. So in this example, the features of nodes $1, 2, 3,$ and 4 would be $1, 4, 1,$ and 1 respectively.
Another convenient way to represent the graph is an adjacency matrix.

Definition 2.3.2 (Adjacency Matrix)**.** An adjacency matrix $\mathcal{A}$ of a graph $(\mathcal{V}, \mathcal{E})$ is in the set $\{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$. In order to do this, the nodes in the graph must be ordered such that every node is assigned a unique value in $\{0, 1, ...|\mathcal{V}|\}$. Then,

$$\mathcal{A}_{ij} = \begin{cases} 0 & (i,j) \notin \mathcal{E} \\ 1 & (i,j) \in \mathcal{E} \end{cases}$$

where i, j are the integer values associated to each node.

In the previous cross example, we would have the adjacency matrix $\mathcal{A} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$.
Note that in our definition of a graph, we are solely dealing with *undirected* graphs, where edges are unordered tuples. In undirected graphs, the adjacency matrix is always symmetrical as the binary relation of edges can be seen to be symmetrical.
Learning and gaining insights from graph-represented data remains a relevant and important class of problem to solve in today's world. Many different types of network problems exist in our world that require inference and analysis. For example, given a popular social media platform such as Twitter, one might be interested in the political polarization of its users over time. Key pieces of information such as text sentiment of its users, who follows whom, and username information might be just some of the data relevant to analyse, just like the housing factors in our last example. Represented as a mathematical graph under our definition, we may represent our users as nodes in the graph, with features being any combination of the datum mentioned above. Given this graph, and all of the associated information, how would one use deep learning techniques to provide answers? This is the question we will be answering in the following section, as an introduction to the structure of GNNs and some of their fundamental properties.
For a detailed reference over different graph neural network architectures, interested reader can see Hamilton et al. 2020 (8).

2.3.1 Graph Neural Networks

The goal of a graph neural network is to infer information on graphs. For deep learning scientists, the best way to do this is to create a *representation* of the graph and its nodes. The identifying approach of GNNs that most literature focuses on is called message passing. In later sections, a second approach (9) we will refer to as the global basis method will also be discussed. For the sake of simplicity and consistency, throughout this work we will assume a popular benchmark task for GNNs: node classification. However, these methods do apply and work for all other types of graph inference tasks as well. The notation and standard definitions used

in this section are mostly taken from Hamilton's text Graph Representation Learning (8).
Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and labels y, the task is to learn a function:

$$f(\mathcal{G}, v) = z_v$$

for all nodes $v \in \mathcal{V}$ where z_v is the target output of the function for the given node in the context of the graph. Since this is a classification task, the labels are one-hot encodings of the c classes, i.e. $y_v \in \mathbb{Z}^c$ and $y_i = 1, y_{j \neq i} = 0 \leftrightarrow$ node v is in class i. y and z are indexed by and represent nodes in the graph, and the loss function here is defined as the negative log-likelihood of the softmax function applied to the target and multiplied with the labels. In other words,

$$\mathcal{L} = \sum_{u \in \mathcal{V}_{train}} -\text{log}(\text{softmax}(z_u, y_u))$$

where the softmax function is defined as:

$$\text{softmax}(z_u, y_u) = \sum_{i=1}^{c} y_u[i] \frac{\exp(z_i)}{\sum_{j=1}^{c} \exp(z_j)}$$

Using this loss, weights are adjusted in backpropagation according to the model's paradigm, which will be discussed below. Note however that loss functions do not have to be this, and the softmax function is simply chosen here due to its desireable property of outputting a vector in $[0, 1]^c$.
A key attribute when it comes to learning over graph structured data however, is *permutation invariance/equivariance*. A naive approach to graph learning may be to simply feed the vectorized adjacency matrix of the graph into an RNN, or some similar deep learning model that trains on sequences. But by doing so, one would be implcitly applying an order to the unordered set information contained in graphs. Indeed, this implies that if the graph was entered into the training model exactly the same as before, only that a given node with numbering 1 were to be numbered 2 instead and another node would be given number 1, the results would be different. This is a situation we would like to avoid, where the function f is not well-defined on graphs.

Definition 2.3.3 (Permutation Invariance/Equivariance)**.** A function f acting on a graph $\mathcal{G}$'s adjacency matrix $\mathcal{A}$ is defined as *permutation equivariant* if for all permutation matrices $P \in \text{Mat}_{|\mathcal{V}|}$,

$$f(P\mathcal{A}P^T) = Pf(\mathcal{A})$$

And f is *permutation invariant* if

$$f(P\mathcal{A}P^T) = f(A)$$

2.4 Current Approaches to GNNs

2.4.1 Message Passing

Message passing is an algorithm designed to aggregate and update node embeddings within a graph. It is also the most popular framework for deep learning on graphs, and the main topic of contribution of this paper.
Given a graph $\mathcal{G}$, the essential algorithm behind message passing is as follows (taken from multiple sections in Hamilton 2020 (8) and placed in table form 1. We assume the same node classifcation task mentioned in subsection 2.3.1.

Algorithm 1 Message Passing for Node Classification, Shallow

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, node features $x_u \in \mathbb{R}^d$, labels $y_u \in \{e_0, ..., e_c\} \forall u \in \mathcal{V}$, where e_i are one-hot encodings of c classes of nodes, $\sigma : \mathbb{R} \to \mathbb{R}$ a nonlinear non polynomial activation function, N the dimension of the hidden layer
Output: $z_u \in \mathbb{R}^c$, the output for each node in terms of probabilities

$W^{(1)}_{self}, W^{(1)}_{neigh} \in \mathbb{R}^{N \times d}$ and $b^{(1)} \in \mathbb{R}^N$ initialized using an initialization algorithm
$W^{(2)}_{self}, W^{(2)}_{neigh} \in \mathbb{R}^{c \times N}$ and $b^{(2)} \in \mathbb{R}^c$ initialized using an initialization algorithm
for $u \in \mathcal{V}$ **do**
    $h^{(0)}_u \leftarrow x_u$
end for
for $u \in \mathcal{V}$ **do**
    $s \leftarrow 0$
    for $v \in \mathcal{V}$ such that $(u, v) \in \mathcal{E}$ **do**
        $s \leftarrow s + h^{(0)}_v$
    end for
    $h^{(1)}_u \leftarrow \sigma(W^{(1)}_{self} h^{(0)}_u + W^{(1)}_{neigh} s + b^{(1)})$
end for
for $u \in \mathcal{V}$ **do**
    $s \leftarrow 0$
    for $v \in \mathcal{V}$ such that $(u, v) \in \mathcal{E}$ **do**
        $s \leftarrow s + h^{(1)}_v$
    end for
    $h^{(2)}_u \leftarrow W^{(2)}_{self} h^{(1)}_u + W^{(2)}_{neigh} s + b^{(2)}$
end for
for $u \in \mathcal{V}$ **do**
    $z_u \leftarrow h^{(2)}_u$
end for
$\mathcal{L} \leftarrow \sum_{u \in \mathcal{V}} -\log(\text{softmax}(z_u, y_u))$
Backpropagate loss $\mathcal{L}$ through $W^{(1)}_{self}, W^{(1)}_{neigh}, W^{(2)}_{self}, W^{(2)}_{neigh}, b^{(1)}, b^{(2)}$ with Stochastic Gradient Descent
Repeat steps 3-24 for any desired number of epochs.
return z_u

Assign to each node u in $\mathcal{G}$ a hidden node embedding h_u. In the first epoch, the value of h_u are the node feature values present at the nodes. The algorithm works in two steps, `UPDATE` and `AGGREGATE`. In the `AGGREGATE` step, each node u generates a set $\mathcal{N}(u)$, the neighbourhood of u where

$$\mathcal{N}(u) = v \forall v \text{ s.t. } (u, v) \in \mathcal{E}$$

Then, some symmetric function, in the standard case as in the algorithm simply the sum function, is applied to the neighbors to create a representation in the dimension of h_u for the neighbors. In the `UPDATE` step, the representation of the neighbors and the representation of the node is passed into a neural network, creating new representations of each node. This process is a single "layer" in the message passing schema. Repeating this any number of times, we end up with the final representation, in the form of what the task at hand requires. For the node classification case, the final representation is a probability distribution over the potential classes. The advantage of this representation is that we can also track the confidence of the model's belief that a node belongs to a particular class as well. This representation for the nodes gets contrasted with the given labels by some loss function and has this loss propagated through each layer, adjusting the weights and biases through stochastic gradient descent.
In one line, the following is the update at $k+1$ time in action(from Hamilton 2020 (8)):

$$h_u^{k+1} = \texttt{UPDATE}^k(h_u^k, \texttt{AGGREGATE}^k(h_v^k, \forall v \in \mathcal{N}(u))) = \texttt{UPDATE}^k(h_u^k, m_{\mathcal{N}(u)}^k)$$

where $m_{\mathcal{N}(u)}^k$ is the message that is aggregated from u's graph neighbourhood. In the standard message passing model, `UPDATE` is an MLP or *multi-layer perceptron* (neural network), and `AGGREGATE` is a simple sum, mean, or max function. In the case where it is a sum function, the equation is:

$$h_u^{(k)} = \sigma(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} h_v^{(k-1)} + b^{(k)})$$

For further details, refer to Algorithm 1.

2.4.2 Expressivity

A key aspect of graph representational learning is the expressiveness of the representations that it learns. Ideally, graphs that are not equivalent should never be considered equivalent in the latent embeddings of the nodes and the graph trained by message passing.

Definition 2.4.1 (Multiset, Xu)**.** A *multiset* is a generalized concept of a set that allows multiple instances of its elements. It is a 2-tuple $X = (S, m)$ where S is the underlying set of X that is formed from its distinct elements, and $m : S \to \mathbb{N}_{\geq 1}$ maps each element to its multiplicity in the set. X will be used to denote the space of all multisets.

Using our definition of neighbourhoods in graphs defined in previous sections, a neighbourhood may be considered a multiset in terms of the collection of its feature values. Furthermore, it is noted that graph structures are essentially collections of connected neighbourhoods. A change in one neighbourhood of a graph by node or edge deletion changes the graph itself. Thus, the question of GNN expressivity can be considered as the discriminating capabilities of the GNN acting on multisets. More specifically, its `AGGREGATE` function.

Examples of Aggregation Functions

Aggregation functions are perhaps the most heavily studied aspect of GNN's, and the topic of interest in this paper. Algorithm 1 is the most standard aggregation function definition. There have been several ideas to improve the `AGGREGATE` operator, starting with normalization in Kipf and Welling's Graph Convolutional Network (GCN) (10).

$$m_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \frac{h_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}$$

Normalization rescales the messages aggregated from the neighbours so that information received from high degree nodes receive less weight in the final calculation. This worked well for the evaluation setting in Welling 2016 (10), as the task benchmark was focused on finding community structure within citation networks of literature. This scaling ensures that works with high amounts of citation and that had high amounts of application in numerous unconnected fields receive less weight when comparing relevance of works.
Another idea is that of neighbourhood attention, used in the Graph Attention Network (11). The GAT uses attention weights on each of a nodes neighbours to produce a weighted sum:

$$m_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}u} \alpha_{(u,v)} h_v$$

where $\alpha_{(u,v)}$ denotes a weight of an edge between nodes u and v. These weights are also computed by gradient descent. This construction runs parallel to the aggregation function however, and can be added on as an additional boost to flexibility in representation. However, neither of these approaches are fully expressive.

The Weisfeiler-Lehman Test

In "How powerful are graph networks" by Xu et al. 2018 (12), Xu et al. characterize the expressiveness of graph neural networks following the message passing paradigm by proving it is no more powerful at distinguishing two non-isomorphic graphs as the Weisfeiler-Lehman graph isomorphism test.
The Weisfeiler-Lehman test for graph isomorphism, or WL test, is an algorithm designed to approximate the solution to the graph isomorphism problem. It does not solve the problem, eg. by perfectly determining graph isomorphism, but succeeds

in distinguishing a wide variety of classes of graphs. The graph isomorphism problem itself, that of determining whether two graphs are isomorphic, has no known polynomial time solution.

Theorem 2.4.1 (GNNs are no more powerful than WL test, Xu (12))**.** *Let G_1 and G_2 be two non-isomorphic graphs. If a graph neural network $\mathcal{A} : \mathcal{G} \to \mathbb{R}^d$ using the message passing algorithm maps G_1 and G_2 to different embeddings, the Weisfieiler-Lehman graph isomorphism test also decides that G_1 and G_2 are not isomorphic.*

In other deep learning models, such a characteristic may be analogous to "universality", a model's ability to approximate any continuous function to any degree of uncertainty. In this case, the analogy may be seen in that while message passing GNNs may not be able to approximate any graph function "continuously" (read: all infinitesimal changes in graphs change the output of the model respectively), it can do so up to the accuracy of an isomorphism test algorithm.
Given the boundary on expressivity from the WL test, a natural question is which GNNs succeed in reaching this boundary, if any. In Xu et al. 2018 (12), the author proves that, given that the underlying set for the features of the nodes is countably infinite, so long as the `UPDATE` and `AGGREGATE` functions are injective with respect to the multiset input, the neural model will be equivalent in expressiveness to the WL test.

Theorem 2.4.2 (Condition on maximal expressiveness, Xu (12))**.** *Let $\mathcal{A} : \mathcal{G} \to \mathbb{R}^d$ be a GNN. With a sufficient number of GNN layers, $\mathcal{A}$ maps any graphs G_1 and G_2 that the Weisfeiler-Lehman test of isomorphism decides as non-isomorphic, to different embeddings if the following holds: $\mathcal{A}$ aggregates and updates node features recursively with*

$$h_v^{(k)} = \phi(h_v^{(k-1)}, f(h_u^{(k-1)} : u \in \mathcal{N}(u)))$$

where the functions f and ϕ are injective.

For example, consider the function $f(x) = N^{-Z(x)}$ where $N \in \mathbb{N}$ and $Z : \mathcal{X} \to \mathbb{N}$. Because $\mathcal{X}$ is assumed to be countable, and furthermore that the cardinality of multisets are bounded, Z may be taken to be an injective function on the multiset and N can be any natural number bigger than the size of the multiset. Then, f is constructed as an injective function. A model, called the "Graph Isomorphism Network (GIN)", that satisfies these constraints is also provided.

$$h_v^{(k)} = \text{MLP}_U^{(k)}((1 + \epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} \text{MLP}_A^{(k)}(h_u^{(k-1)})) \tag{2.2}$$

However, this requirement that the underlying set be countable is too restrictive, and is clearly not the setting for most real world applications where our feature set is the real numbers, and so we turn to the results found in Corso et al.'s "Principal Neighbourhood Aggregation" (2020) (13).

Definition 2.4.2 (Aggregator, Corso)**.** An *aggregator* is a continuous function $f : X \to \mathbb{R}^d$ that acts on a multiset X and returns an aggregated statistic, in the real space.

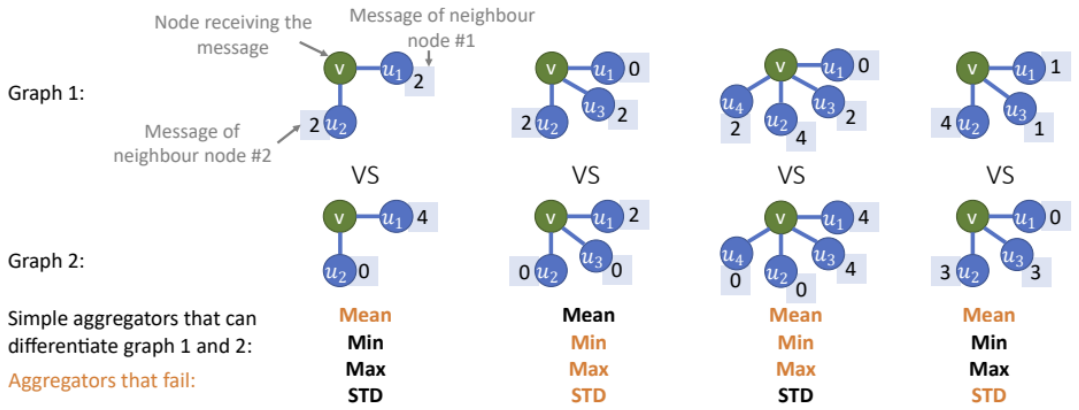

Figure 2.1: A diagram of where some aggregators fail, taken from Corso et al. 2020 (13)

Note in this definition, we are implicitly assuming permutation invariance in the aggregator function, as a function that acts on an *unordered* multiset. An example of where aggregators may fail to distinguish different neighbourhoods may be seen in Fig. 2.1.

Theorem 2.4.3 (Number of aggregators needed, Corso (13))**.** *In order to discriminate between multisets of size n whose underlying set is* $\mathbb{R}$*, at least n aggregators are needed.*

This theorem implies that any `AGGREGATE` function, if it needs to be injective against the set of all multisets of fixed size on $\mathbb{R}$, needs to be comprised of at least n distinct aggregators.
We note here that this result is a result on the worst case scenario of the GNN model. A general sketch of the proof of theorem 2.4.3 is as follows. Assume by contradiction that it is possible to discriminate between all multisets of size n using only $n-1$ aggregators. Define $f : S \to \mathbb{R}^{n-1}$ as the function mapping from the space of multisets of size n, $S \subseteq \mathbb{R}^n$ to $\mathbb{R}^{n-1}$. As a function from a subset of $\mathbb{R}^n$ to $\mathbb{R}^{n-1}$, the Borsuk-Ulam theorem from geometry states there must be two distinct, nonzero points in S that map to the same point in R^{n-1}. This implies that f cannot be injective at at least one point in $\mathbb{R}^{n-1}$.
Given this, however, it is clear that it is possible that at *most* points, less than n aggregators are sufficient in the expressivity sense. Indeed, this can be seen in the moderate success of older GNN models such as the GCN (10), and the GIN from above, which only rely on a single aggregator.
Corso et al. present a GNN model called the Principal Neighbourhood Aggregation (PNA) method, greatly expanding the number of aggregator functions used in an attempt to create more expressivity. A diagram may be seen in Fig. 2.2. In the PNA model, 4 aggregators are applied at the aggregation step, and then 3 scaling functions are applied to each aggregator for a total of 12 aggregation functions. These aggregation functions are then added as input into a neural network to conclude the `AGGREGATE` step. They also note that if necessary, in graphs where the maximum

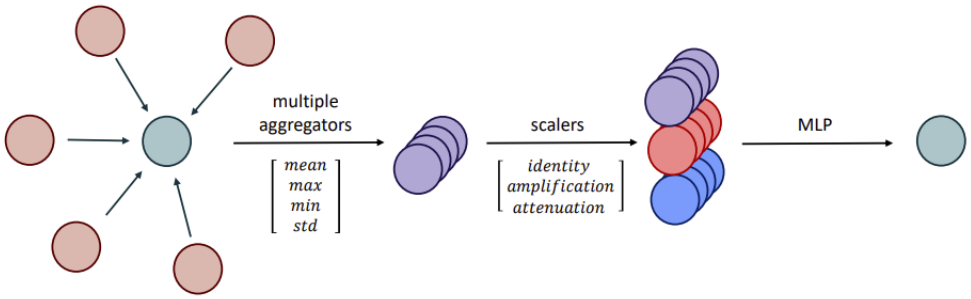

Figure 2.2: Principle Neighbourhood Aggregation (13)

degree at any node (number of connections) is greater than 12, more moments may be added as bonus aggregators.

$$h_v^{(k)} = \text{MLP}_U(h_v^{(k-1)}, \bigoplus_{u \in \mathcal{N}(u)}^{\star} \text{MLP}_A(h_u^{(k-1)})) \tag{2.3}$$

Here, we write $\bigoplus^{\star}$ to denote the PNA's combination of its many aggregators.

Definition 2.4.3. The *moments* of a multiset X are $M_n(X)$, where

$$M_x(X) = \sqrt[n]{\mathbb{E}[(X-\mu)^n]}, n \geq 1$$

where μ is defined as the mean of the multiset.
Justification of the choice of aggregators used in Corso are restricted to arguments of numerical stability. Indeed, they note many other aggregators may be used such as the power sums and the elementary symmetric sums, so long as they are continuous and symmetric. The paper makes no claim about the optimality of such a choice of aggregators, only empirical evidence of its success. Our solution will expand the scope of aggregators used in a much more flexible approach, detailed in chapter 3.

2.4.3 Global Basis Approach

In the paper by Maron et al "Invariant and Equivariant Graph Networks", a different approach to graph neural networks is presented by constructing an orthogonal basis for linear, permutation invariant and equivariant layers in a GNN.
The construction of the data input is as follows: Given a (hyper-)graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of k dimensions, the following tensor of the graph is the data. The number of nodes of the graph is s. Given that node-level information is n-dimensional, and edge-information is e-dimensional, the entire data $D \in \mathbb{R}^{s^k * m}$ of the graph can be constructed as:

$$D_i = \begin{cases} n_q & i = (q, q, ...) \\ e_i & otherwise \end{cases}$$

where m = max(n, k), $n_q \in \mathcal{V}$, and $e_i \in \mathcal{E}$ is the edge level data at the index $i \in I$(index set of the graph). The node information is simply aligned along the

identity of the tensor. For simplicity, m is assumed to be 1 but it is shown in the paper that all of the later constructions also hold in the more general case. Note that in the two-dimensional case, with no node or edge features this definition devolves into the definition of an adjacency matrix, as in definition 2.3.2.
Now that the data is defined, the learning task is the same as described in the sections above. We characterise two linear operators, one equivariant and one invariant. It is clear that multiple equivariant and invariant layers applied sequentially in a neural network create an overall equivariant or invariant function. Let P be a permutation matrix representing a permutation p. Let a general invariant linear operator $L : \mathbb{R}^{s^k} \to \mathbb{R}$ be order invariant if

$$P^{\otimes k}\text{vec}(L) = \text{vec}(L) \tag{2.4}$$

where $\otimes$ is the tensor product and vec is the vectorizing operation (ordering the data tensor by a natural ordering). Similarly, let a general equivariant linear operator $L : \mathbb{R}^{s^k} \to \mathbb{R}^{s^k}$ be equivariant if

$$P^{\otimes 2k}\text{vec}(L) = \text{vec}(L) \tag{2.5}$$

Furthermore, given a tensor $X \in \mathbb{R}^{s^k}$, let an index space on X be $[s]^k$, the set of all k tuples of integers between 1 and s. Using these indices, we define an equivalence relation $\sim$ where for $a, b \in [s]^k$, $a \sim b \leftrightarrow (a_i = a_j \leftrightarrow b_i = b_j)$. This can easily be confirmed as an equivalence relation. The equivalence classes which this generates, denoted as $[s]^k/\sim$, can be represented by a unique partition of the set $[k]$. For example, if k = 2 we have 2 equivalence classes $\gamma_1 = \{\{1\}, \{2\}\}$ and $\gamma_2 = \{\{1, 2\}\}$. γ_1 represents all indexes (i, j) where $i \neq j$, while γ_2 represents all where $i = j$.
Then, we construct a bijection relation where for each equivalence class γ, we define an order-k tensor $B^\gamma \in \mathbb{R}^{n^l}$.

$$B^\gamma_a = \begin{cases} 1 & a \in \gamma \\ 0 & otherwise \end{cases}$$

In our case above, B^{γ_2} would simply be the identity tensor.
These B^γ form an orthogonal basis to the solution set of equations of 2.3. As an informal proof, note that being a solution to (2.3) is equivalent to being constant on equivalence classes–i.e., all elements in a tensor X in the same equivalence class of indices are equal. The identity tensor, for example, is constant on equivalence classes because $I_a = 1$ if $a \in \gamma_2$ and $I_a = 0$ if $a \in \gamma_1$. Since the equality relation $\sim$ is preserved under permutations of the underlying tensor, as long as all the elements in each equivalence class of indices are equal all permutations will satisfy (2.3). It can then be seen that all tensors constant on equivalence classes can be written as a linear combination of elements from B^γ, which are just indicators of the class.
Thus, the dimension of the solution set is the number of partitions of the dimension, or $b(k)$. And as the dimension of the tensors in the equivariant case is $\mathbb{R}^{s^{2k}}$, it is $b(2k)$ respectively. As an example, in the common two dimensional case of a normal graph with an adjacency matrix, the dimension of the solution set of invariant linear tensors is 2 as $b(2) = 2$, and the solution set of the equivariant tensors is 15 as $b(4) = 15$. This is summarised in Theorem 1 in the paper.

In terms of biases, we can extend the previous analysis as well. First, it is noted that any constant layer $\mathbb{R}^{s^k} \to \mathbb{R}$ must be invariant by definition. Second, any equivariant layer $L : \mathbb{R}^{s^k} \to \mathbb{R}^{s^k}$ must satisfy

$$P^{\otimes k}\text{vec}(C) = \text{vec}(C)$$

So, according to the same process above any constant equivariant layer must have the same dimensionality of invariant linear layers–that is, b(k).
Through some more general matrix manipulations, Maron also shows equivalent results for when we extend to multi-dimensional values for node and edge-level info. However, the derivation is involved and will be omitted.
The construction that Maron presents is much more closely in line with typical neural network constructions. For one, it does not operate its layers element-wise as in the message passing construction. The entire data structure is added as input, and the boundary limitations of being permutation invariant restrict the form of the network layers sufficiently to create a finite set of basis elements to apply a network to. Indeed, the global basis method sidesteps the boundary of the WL test on expressiveness, and in fact has been shown to be universal by Keriven (2019) (14).
However the drawbacks of such an approach are also evident in the construction. By design, given a graph of two dimensions, while the standard message passing approach is $\mathcal{O}(|\mathcal{V}|\mathcal{E})$ (where $\mathcal{E}$ is the average number of neighbours per node) in time complexity, the global basis approach is $\mathcal{O}(|\mathcal{V}|^2)$, suggesting an inefficiency to this method. In fact, the complexity is most comparable to operations on Deep Sets (3), networks that process completely unconnected sets of nodes, implying the underlying graph structure is not efficiently parameterised. The worst case performance can be observed in the case of sparse networks, where most of the adjacency matrix is empty.

2.5 Invariant Theory and Machine Learning

This section of the report will detail some basics of invariant theory in the context of this problem as well as its application to the problem at hand. The general approach covering CNNs (and in the future, other architectures) is the current topic of research of Victoria Klein, Jeroen Lamb, and Kevin Webster who are supervising this project. Much of the following text is cited from the early stage assessment of Victoria's thesis [not published], as well as results from Yarotsky et al. 2022 (15).
Given our learning task on graphs, and the permutation invariant functions we wish to learn on its nodes, in order to characterize them properly we need a tool to refine our definition of permutation invariance. For this, we turn to group representations and the symmetric group.

Definition 2.5.1 (Group Representation)**.** Let G be a group written multiplicatively with its binary operation and identity e_G. A *representation* of a group G is a pair (V, ρ), where V is a vector space, $GL(V)$ is the general linear group on V (the group of invertible linear transformations from V to itself), and $\rho : G \to GL(\text{V})$ is a group homomorphism.

Representations of a group may also be defined as invertible matrices representing group elements acting on a chosen vector space. The distinction between the two definitions is simply a choice of basis for the general linear group. For the purposes of this paper, as we are focused on applications to machine learning, we will be working with finite groups and finite-dimensional representations over $\mathbb{R}$, but they may be defined more generally.
Let us now define what it means for a function to be invariant with respect to a group action.

Definition 2.5.2 (Function invariance/equivariance)**.** Let G be a group and let (V, ρ) be a representation. Then a map $f : V \to \mathbb{R}$ is called *G-invariant* (or simply *invariant*) if

$$f(\rho(g)x) = f(x) \text{ for all } g \in G \text{ and } x \in V$$

Furthermore, let (U, μ) be another representation of G. Then a map $f : V \to U$ is *G-equivariant*, or more precisely *($\rho : \mu$) equivariant*, if

$$f(\rho(g^{-1})x) = \mu(g)f(x) \text{ for all } g \in G \text{ and } x \in V$$

A *polynomial invariant* is simply a G-invariant polynomial function. An equivariant map $f : V \to U$ is *polynomial equivariant* if $l \circ f$ is a polynomial for any linear functional $l : U \to \mathbb{R}$. We now look at a classical result from Hilbert on polynomial invariance.

Theorem 2.5.1 (Hilbert's Finiteness Theorem (16), (17))**.** *Let G be a compact group and (V, ρ) be a representation of G in the space V. Then there exist finitely many polynomial invariants $f_1, ...f_{N_{inv}} : V \to \mathbb{R}$ such that any polynomial invariant $r : V \to \mathbb{R}$ can be expressed as:*

$$r(x) = \tilde{r}(f_1(x), ..., f_{N_{inv}}(x))$$

with some polynomial $\tilde{r}$ of N_{inv} variables.

Furthermore, we can extend Hilbert's Theorem to the equivariant case as well.

Theorem 2.5.2 (Hilbert's Finiteness Theorem (equivariant)(16), (17))**.** *Let G be a compact group and (V, ρ), (U, μ) be representations of G in the spaces V, U. Then there exist finitely many polynomial invariants $f_1, ...f_{N_{inv}} : V \to \mathbb{R}$ and polynomial equivariants $g_1, ..., g_{N_{eq}} : V \to U$ such that any polynomial equivariant $r_{sym} : V \to U$ can be expressed as:*

$$r_{sym}(x) = \sum_{m=1}^{N_{eq}} g_m(x)\tilde{r}_m(f_1(x), ..., f_{N_{inv}}(x))$$

with some polynomials $\tilde{r}_m$ of N_{inv} variables.

These finite sets of polynomial equivariants and invariants resulting from Hilbert's theorem will be denoted as the *generating sets*. In other works [see Victoria ESA], they are also known as the fundamental invariants and equivariants. Combining this theorem with Theorem 2.2.1 and using the density of the polynomials in the space of continuous functions, we have the result from Yarotsky et al. 2022 (15):

Theorem 2.5.3 (Yarotsky). *Let G be a compact group, V a finite-dimensional representation of G, and $f_1, ...f_{N_{inv}} : V \to \mathbb{R}$ the generating set of polynomial invariants on V (existing by Hilbert's theorem). Then any continuous invariant map $f : V \to \mathbb{R}$ can be approximated by invariant maps $\hat{f} : V \to \mathbb{R}$ of the form*

$$\hat{f}(x_1, ..., x_d) = \sum_{n=1}^{N} c_n \sigma(\sum_{s=1}^{N_{inv}} w_{ns} f_s(x) + h_n)$$

with some parameter N and coefficients c_n, w_{ns}, and h_n. So, with our input $x \in \mathbb{R}^d$ transformed into $f_{inv}(x) \in \mathbb{R}^{N_{inv}}$ via f_s:

$$\hat{f}(x) = \mathcal{L}_c(\sigma(\mathcal{L}(f_{inv}(x)))) \tag{2.6}$$

where $\mathcal{L} : \mathbb{R}^{N_{inv}} \to \mathbb{R}^n$ and $\mathcal{L}_c : \mathbb{R}^N \to \mathbb{R}$.

And equivalently the equivariant version of this theorem:

Theorem 2.5.4 (Yarotsky, equivariant). *Let G be a compact group, and V and U two finite-dimensional representations of G. Let $f_1, ...f_{N_{inv}} : V \to \mathbb{R}$ a finite generating set of polynomial invariants on V and $g_1, ..., g_{N_{eq}} : V \to U$ be a finite generating set of polynomial equivariants. Then any continuous equivariant map $f : V \to U$ can be approximated by equivariant maps $\hat{f} : V \to U$ of the form*

$$\hat{f}(x) = \sum_{n=1}^{N} \sum_{m=1}^{N_{eq}} c_{nm} g_m(x) \sigma(\sum_{s=1}^{N_{inv}} w_{mns} f_s(x) + h_{mn})$$

with some parameter N and coefficients c_n, w_{ns}, and h_n. So, with our input $x \in \mathbb{R}^d$ transformed into $f_{inv}(x) \in \mathbb{R}^{N_{inv}}$ via f_s:

$$\hat{f}(x) = \mathcal{L}_c(\bigoplus_{m=0}^{N_{eq}} g_m \otimes (\sigma(\mathcal{L}(f_{inv}(x))))_m) \tag{2.7}$$

where $\mathcal{L} : \mathbb{R}^{N_{inv}} \to \mathbb{R}^{N_{eq} \times N}$ and $\mathcal{L}_c : \mathbb{R}^{N_{eq} \times N} \to \mathbb{R}^{\delta}$, where δ is the dimension of U.

It is clear, then, from this we have a suitable construction for a neural network to learn $G - invariant$ and $G - equivariant$ maps for any particular compact group G. The only things we are missing are to calculate the generating sets of fundamental invariants and equivariants for our group of choice, as well as the method to apply this to graph neural networks. We shall show these in the following sections.

2.6 The Symmetric Group

Definition 2.6.1 (Symmetric Group). The *symmetric group* S_n is composed of all bijections from the set $\{1, 2, ..., n\}$ to itself, using composition as its binary operation.

Elements $\pi \in S_n$ are called *permutations*. We will write permutations in cycle notation. Given the cycle $(i, j, k, \cdots, l)$, this means π sends i to j, j to k, and so on until it sends l to i. Any permutation can be expressed in cycle notation, $\pi = (1, 2, 3)(4)(5)$ for example, sends 1 to 2, 2 to 3, and 3 to 1 while fixing 4 and 5. A 1-cycle, or cycle of length 1, is known as a *fixed point*.
In terms of functions invariant to the symmetric group, Zaheer (2018) (3) proves the following:

Theorem 2.6.1 (Functions invariant to S_n, Zaheer)**.** *Let $f : X \to \mathbb{R}$, where X is a set of size n of elements from any compact set in $\mathbb{R}$, be any continuous function invariant to permutations on the order of its n inputs. Then it can be approximated arbitrarily close in the form of*

$$f \approx \rho(\sum_{x \in X} \phi(x))$$

where $\phi(x) : \mathbb{R} \to \mathbb{R}^{n+1}$ and $\rho : \mathbb{R}^{n+1} \to \mathbb{R}$.

However, in Zaheer's construction we see no application to GNNs, as the focus of his study is primarily on completely unordered sets. Furthermore, this is a result in reference to *sets*, where as we are primarily focused on *multisets*.

Chapter 3

Symmetric Aggregation

3.1 The Symmetric Basis

It is stated in section 2.5, that there is a theoretical basis to constructing neural layers capable of approximating any symmetric group-invariant function. Our goal is to now apply this to the `AGGREGATE` function. What is left then is to determine some generating sets of fundamental invariants of S_n. Yarotsky (15) notes that the power series of polynomials up to N variables is a well-known set of generating invariants for S_n.

Definition 3.1.1 (Power sum symmetric polynomials)**.** Given n variables $X_1, ..., X_n$, the *power sum symmetric polynomials* $p_k(X_1, ..., X_n)$ for $k = 1, ..., n$ are defined by

$$p_k(X_1, ..., X_n) = \sum_{i=1}^{n} X_i^k$$

However, Hilbert's theorem (Thm. 2.5.1) does not define a unique finite set of invariants, only the existence of one. There exist computer algebra systems to calculate an exact minimal (in terms of number of elements) set of invariants, and there is an argument to be made about reducing dimensionality (and thus, compute time) of our model working with a smaller set of invariants. Indeed, there also exist other generating invariant sets for the symmetric polynomials, one of which is the elementary symmetric polynomials.

Definition 3.1.2 (Elementary symmetric polynomials)**.** Given n variables $X_1, ..., X_n$, the *elementary symmetric polynomials* $e_k(X_1, ..., X_n)$ for $k = 1, ..., n$ are defined by

$$\begin{aligned} e_1(X_1, ..., X_n) &= \sum_{1 \le j \le n} X_j \\ e_2(X_1, ..., X_n) &= \sum_{1 \le j \le k \le n} X_j X_k \\ &... \\ e_n(X_1, ..., X_n) &= X_1 X_2 ... X_n \end{aligned}$$

This set of invariants is the set used in the fundamental theorem of symmetric polynomials, which states that the elementary symmetric sums are a generating set of invariants for S_n.
The choice of basis to use is up to the implementer, but for this model the power sum polynomials will be used for reasons that will be made clear.

3.2 Symmetric GNN

3.2.1 The Construction

For the model, we will assume the same problem outlined in the background section of node classification. We will now present the *Symmetric Graph Neural Network*, or *SGNN*.
The new message passing step is:

$$h_u^{(k)} = \sigma(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_{\text{neigh}}^{(k)}(W_c \sigma_s(W_s p(\mathcal{N}(u)) + b_s)) + b^{(k)})$$

where $p : \mathbb{R}^{n\times f} \to \mathbb{R}^{d\times f}$ are the first d power sum symmetric polynomials applied row-wise to $\mathcal{N}(u)$, weights $W_s : \mathbb{R}^{d\times f} \to \mathbb{R}^{N\times f}$, $W_c : \mathbb{R}^{N\times f} \to \mathbb{R}^f$ and $b_s \in \mathbb{R}^{N\times f}$. σ_s is a nonpolynomial activation layer. In a more abstracted form,

$$h_u^{(k)} = \text{MLP}_u(h_u^{(k-1)}, \text{MLP}_a(p(\mathcal{N}(u)))$$

The `UPDATE` function remains the same as in the standard message passing framework, while the `AGGREGATE` function is an MLP applied to the power sum symmetric polynomials of the neighbourhood of the node.
Below is the main result of this paper.

Theorem 3.2.1. *The SGNN's* `AGGREGATE` *function is a generalisation of every message passing neural network's* `AGGREGATE` *function, and if d is chosen to be the maximum degree of nodes in the graph, is equal in expressivity to the WL test.*

Proof. By construction, we have seen that the `AGGREGATE` function is nothing more than a collection of aggregators, defined in Def. 2.4.2. As an aggregator is a continuous function invariant to the actions of the symmetric group, we see that our construction approximates any aggregator to any given degree of uncertainty by mimicking the construction of Thm. 2.5.3.
As for the second statement, we note that our model is a generalisation of the PNA model proposed in Corso (13), in that both `AGGREGATE` functions are MLP's applied to a collection of aggregators. By Thm. 2.4.2, we see that a sufficient condition for expressivity equivalent to the WL test is having at least n aggregators, where n is the number of nodes in the neighbourhood of any given node. Taking d to be the maximum degree of the nodes in the graph enforces this, and the conclusion follows. □

Looking at the paper by Corso et al., it may seem quite similar in approach. However, a key difference to note is the application of a neural network to the aggregator

functions, rather than aggregators applied to representations of the nodes already passed through a network. The proof of injectivity in the Corso paper on PNA relies on the inner MLP_A in equation 2.3 learning a very restricted class of injective representations, similar to that of the GIN 2.2. Both GIN and PNA's injectivity proof is valid in the sense that it is possible for the combined MLP's to learn a specific injective function by having the inner MLP_A learn a function that is exponential in input parameters and that the agggregators sum over to produce an injective overall function. However this is clearly an inefficient approach as it necessitates the models to learn over a very large function space (since the width of MLP_A must grow exponentially with the inputs), with only a tiny fraction (minus exponential) of them being valid. Our approach on the other hand, starts out with n different aggregators as a basis, and then feeds them into an MLP. The advantage of this is that there is no hard requirement for the model to learn over the same exponentially-sized function space, while maintaining injectivity by the original Corso proof.

3.2.2 Numeric Stability

A predictable problem with the SGNN model is that of numeric stability. While theoretically the results hold, given that the power sum symmetric polynomials are a basis with elements differing by scales of magnitude the resulting weight matrix corresponding to them W_s will be extremely ill-conditioned. To improve this, a further modification to the basis is made:

Definition 3.2.1 (Norm basis)**.** Given neighbourhood $\mathcal{N}(x) = \{x_1, x_2, ..., x_n\}$, the *norm basis* is ν_i, defined as:

$$\nu_i = \sqrt[i]{x_1^i + x_2^i + ... + x_n^i} = \|x\|_i$$

where $\|x\|_i$ may be understood to be an "L^i norm" on the neighbourhoods of each node.

Theorem 3.2.2. *The original results for the SGNN still hold for the norm basis being used instead of the power sum basis.*

Proof. So long as a set of objects is a generating set for the invariants of S_n, it is a valid basis for the SGNN, by Thm. 2.5.3. Taking polynomials of the norm basis, we can easily recover the original power sum symmetric polynomials by taking each element $\nu_i^i = p_i$. Thus ν_i are a generating set for S_n □

This is also the reason to use the power sums as the initial basis. It simplifies the end construction here as shown.

3.2.3 Efficiency Improvements

As noted in the background, the main reason to set d, the degree of the power sum polynomials, to the maximum degree of the graph is to attain the guarantee of injectivity of the aggregation step. However, as also stated previously this guarantee

is not worth particularly much, given the success of graph networks with only a single aggregator such as GCN and GIN. Given this, a modification to boost efficiency of calculations can be made where we take d to be the average degree of nodes in the graph, or some other summary statistic of degrees, and see how that affects the accuracy of results.

3.3 Limitations

Due to us only approximating compactly supported continuous functions, if the function that we need to approximate is particularly ill-conditioned, such as a step function, it will take more iterations to converge to a continuous approximation of the function. Since compactly supported continuous functions are dense in the space of $L^p[K]$ (Lebesgue p-integrable functions on compact support), it is still possible to approximate a wider variety of functions however it can be shown that the smoother the function is, the quicker convergence will be.

Chapter 4

Evaluation

One problem already mentioned in the efficiency improvement section is the amount of calculations needed to be done to switch from the feature space basis to the norm basis. This, added to the existence of another neural network needed to be calibrated by gradient descent is a constant increase on computational and space complexity on top of the standard message passing NN. If the message passing neural network has time complexity of $\mathcal{O}(|\mathcal{V}||\mathcal{E}|)$, the SGNN has complexity $\mathcal{O}(|\mathcal{V}|(|\mathcal{E}|+|\mathcal{V}|))$, a significant increase. The additional bonus is calculated for worst-case in a scenario where the maximum degree of a node in a fully connected graph is $|\mathcal{V}|$.
A comparison may be made to Maron's global basis construction, which is also on $\mathcal{O}(|\mathcal{V}|^2)$ complexity. However, it is clear to see that while Maron's construction enforces this complexity as a baseline, given its large matrix operations, the SGNN only sees a similar complexity in the worst case scenario of a graph with a node connecting to all other nodes. And given the modification of changing this to the average degree of the graph, while the worst case complexity does not change the amortized complexity decreases yet further to be the same as the MPNN, $\mathcal{O}(|\mathcal{V}||\mathcal{E}|)$.
In terms of actual gain, it is not clear what additional flexibility in the aggregation function offers to a GNN. The limits of expressivity according to the WL test are achieved, as in the case of the PNA model offered by Corso. However the author believes that due to the universal construction of the aggregate function provided in this paper, there is a value in the generalisation of the aggregation step if in nothing else but theory. It will no longer be necessary to consider, for a given graph, whether a mean or max aggregator is appropriate for the aggregator; the SGNN will learn what the appropriate form is.

Chapter 5

Ethical Issues

As this is quite a theoretical project, I would not see any immediate ethical considerations that give pause. The implications of better graph neural networks in particular are incredibly varied and contributing to a deeper understanding of what they are and a general theory behind machine learning is even more so. The main aspects to consider would be in the applications of GNNs, in which case the user is referred to studies on the ethicality of machine learning in general (18), and perhaps as an example the effects of social media marketing (19) as it is one of the key use cases of GNNs (social media networks).

Chapter 6

Conclusions

6.1 Reflection

The algebraic construction of the norm basis for the SGNN provides an elegant and computationally effective form of symmetrization to the aggregation step in GNNs. Utilising the theory of symmetric functions and the universality of neural networks, we have proven a generalisation of a critical step of GNNs with minimal added complexity of calculations. The technique of algebraic approximation via polynomials used in this paper is not novel; however its application to this function of GNNs is.

6.2 Future Works

In terms of implementations, the clear next direct step is to implement the SGNN and compare its performance to state of the art models. This is something the author plans to do immediately, and continue on existing work already done. As the research of GNNs continues into permutation-sensitive over permutation-invariant constructions, it would be interesting to see an extension of SGNNs into utilising the fundamental invariants of a subgroup of S_n, such as the Alternating group A_n. As the literature has shown, the permutation invariant limitation of GNNs has spawned a vast topic of research; however the pool is growing thin. Already we have shown a universal construction in the global basis method, and this paper shows what form message passing networks must take to be permutation invariant. It seems imperative now, that we look at methods to relax this constraint to improve expressiveness yet further while at the same time maintaining a valid model.

Bibliography

[1] Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE. Neural message passing for quantum chemistry. In: International conference on machine learning. PMLR; 2017. p. 1263-72. pages i

[2] Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P. Geometric Deep Learning: Going beyond Euclidean data. IEEE Signal Processing Magazine. 2017;34(4):18-42. pages 1

[3] Zaheer M, Kottur S, Ravanbakhsh S, Poczos B, Salakhutdinov R, Smola A. Deep sets. arXiv preprint arXiv:170306114. 2017. pages 2, 16, 19

[4] LeCun Y, Bengio Y, Hinton G. Deep learning. nature. 2015;521(7553):436-44. pages 4

[5] Pinkus A. Approximation theory of the MLP model in neural networks. Acta numerica. 1999;8:143-95. pages 5

[6] Kelley HJ. Gradient theory of optimal flight paths. Ars Journal. 1960;30(10):947-54. pages 6

[7] Amari Si. Backpropagation and stochastic gradient descent method. Neurocomputing. 1993;5(4-5):185-96. pages 6

[8] Hamilton WL. Graph representation learning. Synthesis Lectures on Artifical Intelligence and Machine Learning. 2020;14(3):1-159. pages 7, 8, 9, 10

[9] Maron H, Ben-Hamu H, Shamir N, Lipman Y. Invariant and equivariant graph networks. arXiv preprint arXiv:181209902. 2018. pages 7

[10] Welling M, Kipf TN. Semi-supervised classification with graph convolutional networks. In: J. International Conference on Learning Representations (ICLR 2017); 2016. . pages 11, 13

[11] Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y. Graph attention networks. arXiv preprint arXiv:171010903. 2017. pages 11

[12] Xu K, Hu W, Leskovec J, Jegelka S. How powerful are graph neural networks? arXiv preprint arXiv:181000826. 2018. pages 11, 12

[13] Corso G, Cavalleri L, Beaini D, Liò P, Veličković P. Principal neighbourhood aggregation for graph nets. Advances in Neural Information Processing Systems. 2020;33:13260-71. pages 12, 13, 14, 21

[14] Keriven N, Peyré G. Universal invariant and equivariant graph neural networks. Advances in Neural Information Processing Systems. 2019;32. pages 16

[15] Yarotsky D. Universal approximations of invariant maps by neural networks. Constructive Approximation. 2022;55(1):407-74. pages 16, 17, 20

[16] Hilbert D. On the theory of algebraic forms. Mathematical annals. 1890;36(4):473-534. pages 17

[17] Hilbert D. Über die vollen Invariantensysteme. In: Algebra· Invariantentheorie· Geometrie. Springer; 1933. p. 287-344. pages 17

[18] Lo Piano S. Ethical principles in machine learning and artificial intelligence: cases from the field and possible ways forward. Humanities and Social Sciences Communications. 2020;7(1):1-7. pages 25

[19] Moreno MA, Goniu N, Moreno PS, Diekema D. Ethics of social media research: common concerns and practical considerations. Cyberpsychology, behavior, and social networking. 2013;16(9):708-13. pages 25