ON HIGHER ORDER GRAPH REPRESENTATION LEARNING

by

Balasubramaniam Srinivasan

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer Science

West Lafayette, Indiana

May 2022

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

To the loving memory of my father

ACKNOWLEDGMENTS

I wouldn't have come to this stage of my PhD without the help of many people, and I would like to thank them all sincerely from the bottom of my heart, for all their contributions. First, I would like to thank my advisor Prof. Bruno Ribeiro. Prof. Ribeiro has always been a supportive advisor and his enthusiasm towards performing fundamental research has left a lasting impression on me. Right from first day of my PhD, he has been very invested in improving my research skills and taught me almost everything that I know about research – asking the right research questions, communicating clearly and good writing practices. Additionally, he taught me how to formalize my ideas and always encouraged me to attend conferences, serve as a reviewer and go for internships — aiding my overall development. I would also like to thank Prof. Ribeiro for his patience with me and most importantly for helping me rekindle my love for math - and for this I will always be grateful.

I would also like to thank my thesis committee members – Profs. Vinayak Rao, Pan Li and Daisuke Kihara for their guidance and suggestions in improving my work. Prof. Rao has been extremely encouraging, right from the first days of my PhD, providing valuable ideas, especially while working on the projects of Janossy Pooling and Relational Pooling. Prof. Li has provided valuable feedback to improve my writing and has always made himself available to discuss different research ideas on graphs. Prof. Kihara has been very helpful while working on the project on conditional invariances and provided valuable domain specific insights to help improve my models. Beyond my thesis committee, I would like to thank Prof. Jennifer Neville for the discussions and her ideas while working on the project on knowledge graphs. I would also like to thank Prof. Kiril Datchev who taught me differential geometry and graciously made himself available to discuss ideas outside class hours, even while I was only auditing the course. I must also thank Profs. Leo Porter and Fragkiskos Malliaros (from my time at UC San Diego), Dr Satish Raghunath (from my time at Salesforce) and Dr. Suresh Srinivasan (from my time at ARM), who encouraged me to pursue my PhD and also introduced me to graph representation learning.

I was fortunate to go for internships at Amazon during my PhD and both my internships at Amazon were extremely rewarding. Prof. George Karypis was very polite and approachable, and apart from providing thoughtful ideas and suggestions in my projects, made me feel at home at Amazon. I am extremely delighted with the prospect of working with Prof. Karypis again, in the next phase of my career. I would also like to thank my other collaborators at Amazon - Da Zheng, Vassilis Ioannidis and Soji Adeshina.

I would also like to thank my lab mates at Purdue - with whom I have built friendships to last a lifetime. Ryan Murphy, my collaborator during my first two works at Purdue, mentored me during the first year of my PhD and imbibed in me, the work ethic required of a PhD student. I would like to thank S. Chandra Mouli, Mayank Kakodkar and Leonardo Teixeira - for our weekly math reading meetings (and I hope the meetings continue, long after we complete our PhDs), for making themselves always available when I wanted to discuss research or brainstorm ideas, for proof reading all my paper submissions, for listening and critiquing my practice talks and being a source of support and motivation. I would also like to thank Beatrice Bevilacqua, Leonardo Cotta, Jason Meng, Jianfei Gao, Josue, Shishang Wu and Yangze Zhou for all our thoughtful discussions. Additionally, I would also like to thank Susheel Suresh for collaborating with me on the knowledge graphs project. My roommates over the past few years at Purdue – Vivekanandan, Suraj and Ravishankar (who was also my Physics teacher) treated me as family and were extremely helpful during my times of deadlines. They made my time during graduate school fulfilling and enjoyable, and for this I will always be thankful to them. I would also like to offer my thanks to the administrative staff (Lacey Siefers and Monica Shively) for patiently answering all my questions, helping me with a lot of documentation and travel procedures.

Going further back, I would like to thank Prof. S.K. Nandy, my undergraduate thesis advisor – who introduced me to research and my high school math teacher Mrs. Vijayalakshmi Raman who has always been a well wisher of mine. I would also like to thank my friends from my undergraduate days – Vignesh Balaji and Kaushik Raj, who helped with the PhD application process and provided encouragement during difficult times during my PhD. I would also like to thank my aunt and uncle (Lakshmi and Ganesh), who have been a constant source of support and encouragement during my 10 years of combined undergraduate and graduate study.

Finally, I need to thank the most important people in my life, my parents (Geetha and Srinivasan). They have always put me before themselves and none of this would have ever been possible without them. My mother has always been there for me, especially during my difficult times. Even after losing my father at very young age, she has hid her own disappointments and sadness from me and has always given me complete freedom to pursue my dreams. And while my father is not here with me today, I believe his blessings will always be with me and is the reason for where I am today. This thesis belongs to my parents as much as it does to me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Research on graph representation learning (GRL) has made major strides over the past decade, with widespread applications in domains such as e-commerce, personalization, fraud & abuse, life sciences, and social network analysis. Despite its widespread success, fundamental questions on practices employed in modern day GRL have remained unanswered. Unraveling and advancing two such fundamental questions on the practices in modern day GRL forms the overarching theme of my thesis.

The first part of my thesis deals with the mathematical foundations of GRL. GRL is used to solve tasks such as node classification, link prediction, clustering, graph classification, and so on, albeit with seemingly different frameworks (e.g. Graph neural networks for node/-graph classification, (implicit) matrix factorization for link prediction/ clustering, etc.). The existence of very distinct frameworks for different graph tasks has puzzled researchers and practitioners alike. In my thesis, using group theory, I provide a theoretical blueprint that connects these seemingly different frameworks, bridging methods like matrix factorization and graph neural networks. With this renewed understanding, I then provide guidelines to better realize the full capabilities of these methods in a multitude of tasks.

The second part of my thesis deals with cases where modeling real-world objects as a graph is an oversimplified description of the underlying data. Specifically, I look at two such objects (i) modeling hypergraphs (where edges encompass two or more vertices) and (ii) using GRL for predicting protein properties. Towards (i) hypergraphs, I develop a hypergraph neural network which takes advantage of the inherent sparsity of real world hypergraphs, without unduly sacrificing on its ability to distinguish non isomorphic hypergraphs. The designed hypergraph neural network is then leveraged to learn expressive representations of hyperedges for two tasks, namely hyperedge classification and hyperedge expansion. Experiments show that using our network results in improved performance over the current approach of converting the hypergraph into a dyadic graph and using (dyadic) GRL frameworks. Towards (ii) proteins, I introduce the concept of conditional invariances and leverage it to model the inherent flexibility present in proteins. Using conditional invariances, I provide a new framework for GRL which can capture protein-dependent conformations and

ensures that all viable conformers of a protein obtain the same representation. Experiments show that endowing existing GRL models with my framework shows noticeable improvements on multiple different protein datasets and tasks.

1. INTRODUCTION

Deep learning research has seen a rapid growth in the past decade, both in academia and industry. This can be partly attributed to its ability to be applied to different modalities of data. Moreover, deep learning is able to efficiently leverage web scale datasets to improve performance on unseen data by tapping into the computational power and memory of modern day GPUs and CPUs. Prior to the last 5 years however, deep learning research has largely been restricted to tasks on images (CNN's), text, speech (RNN's) while not many advances were pursued on semi-structured and unstructured data. Specifically, research on graph data (where the input is a graph) only started garnering widespread attention over the past five years [1], [2].

Graphs are particularly special because of their ability to capture information about different entities (vertices/ nodes) as well as model relations (links/ edges) between them. For example, a molecule is a graph where atoms are nodes and two atoms are connected if there is a bond between them, a social network is a graph where people are entities and two people are connected with each other if they are friends and so on. More specifically, graphs are objects

- where the characteristics of an entity may not be completely summarizable by the attributes of just the entity, but require knowing its connectivity with other entities in the input data, i.e. the input data is longer *i.i.d.*,
- where an entity can have connections (edges) with an arbitrary number of other entities,
- with the property that relabeling the nodes/ permuting the adjacency matrix leaves the underlying data unchanged (permutation symmetry).

An input graph G is typically represented as a tuple $(V, E, \boldsymbol{X}_v, \boldsymbol{X}_\mathrm{e})$ where V is the vertex set, $E \subseteq V \times V$ denotes the edge set and $\boldsymbol{X}_v, \boldsymbol{X}_\mathrm{e}$ are the features associated with the nodes and edges respectively.

1.1 An Overview of Graph Representation Learning

Graph representation learning, as the name suggests, entails learning low dimensional representations of graphs which captures important properties about the nodes, links (edges) and the graph as a whole. It finds a plethora of applications in domains such as e-commerce, personalization, fraud and abuse, life science and social network analysis.

Note: When I say graphs, I refer to dyadic graphs where the edges are restricted to being incident on two nodes. In the case where this restriction isn't imposed, I refer to them as hypergraphs.

Tasks on graphs have primarily been categorized as

- *Node level tasks* - which involves learning low dimensional representations of nodes with the goal to classify nodes (*node classification*))/ predict real valued properties associated with individual nodes.
- *Edge level tasks* - which involves predicting the existence/ non-existence (or the properties) of edges between any two entities in the graph and commonly finds applications in knowledge graphs, recommender systems, etc.
- *Graph level tasks* - which involves classifying/ predicting real valued properties of the graph as a whole (e.g. predicting molecular properties).

By and large, graph representation learning frameworks for the above tasks have been broadly categorized into two frameworks, those which learn (a) *structural node representations* aka frameworks which learn permutation invariant representations of nodes where isomorphic nodes obtain identical representations (e.g. graph neural networks) and (b)*positional node embeddings* where homophily usually dictates the embedding that a node receives (e.g. ([1], [3] and other low rank matrix approximations, factor analysis, etc [4]–[6])).

Graph neural networks [2], [7], [8] or frameworks which learn *structural node representations* have been the preferred technique to perform node level tasks, whereas *positional node embedding techniques* have been employed for tasks such as link prediction and clustering. For graph level tasks, graph neural networks [9], [10] have again been employed — where the representation of the graph is obtained by using a permutation invariant pooling function over the individual node representations.

1.2 Contributions

Despite its widespread success, two fundamental questions on practices employed in modern day graph representation learning have remained unanswered. I briefly describe them and provide my solutions below:

1. The predominant downstream tasks associated with graph representation learning are node classification, link prediction, clustering and graph classification. These tasks have leveraged seemingly different learning frameworks to achieve their objectives. For instance, graph neural networks (and more generally, structural representation methods) are primarily used for node and graph classification, while (implicit) matrix factorization methods (and more generally, positional embedding methods) are primarily used for link prediction and clustering tasks. Are these learning frameworks (and correspondingly, structural representations and positional embeddings) fundamentally different? Can they perform each other's tasks?

   In my thesis I provide a theoretical blueprint that connects positional node embeddings and structural graph representations, bridging methods like matrix factorization and graph neural networks. Using tools from group theory, I show that the relationship between structural node representations and positional node embeddings is analogous to the statistical relation between distributions and their samples. With this renewed understanding, my work provides guidelines to better realize the full capabilities of these methods in a multitude of tasks.

2. In many real world scenarios, modeling the underlying data as a graph can be an over-simplified description of the data. How do we capture the properties of the data which are lost while using modern day graph representation learning frameworks on the data? Specifically:

(a) Graph representation learning has primarily been focused on solving downstream tasks over dyadic graphs (graphs where edges are restricted to being incident on two nodes/ or self loops). However, in most cases, interactions may occur between more than two entities, and different interactions may occur between a different number of entities(hypergraphs). For example, consider a dataset where politicians are nodes and co-sponsoring a bill is an interaction — note that more than two politicians can co-sponsor the same bill. This raises the question: can we learn representations on hypergraphs for a task where we would like to accurately predict the nature of an unseen bill, given the set of co-sponsors (problem of hyperedge classification)? Additionally, given a partial set of co-sponsors of an unseen bill, can we accurately predict other politicians likely to co-sponsor the bill (problem of hyperedge expansion)?

    Here, I develop a hypergraph neural network which works on real world sparse hypergraphs which have hyperedges of different cardinalities. I provide a framework to learn provably expressive representations of hyperedges for the two tasks (hyperedge classification, hyperedge expansion) which result in improved performance over the current standard approach of converting the hypergraph into a dyadic graph and using (dyadic) graph representation learning frameworks.

(b) The underlying data can exhibit invariances to transformations in addition to the symmetries of graphs. For instance consider a protein modeled as a graph (using its contact map), and the associated task of predicting its binding affinity with a given ligand. Modeling the protein as a graph alone, doesn't capture that the 3D spatial description of the protein (and the ligand jointly) can be rotated, translated as whole, without changing the outcome. Additionally, some proteins have multiple different conformations (3D structure) all of which are not observed in the training data. These different protein conformations, in most cases, also have the exact same binding affinity with the ligand. The question here is, can we make predictions about the protein regardless of the specific conformation present in the dataset or equivalently learn non trivial conformer invariant representations?

Here, I introduce the concept of conditional invariances and provide a framework which can capture protein dependent conformations and ensures that all viable conformers of a protein (including non-isometric transformations) obtain the same representation. Experiments show that endowing existing models with my framework shows noticeable improvements on multiple different protein datasets and tasks.

1.3 Thesis Outline and Previously Published Work

This thesis is organized as follows:

1. In Chapter 2, I use tools from invariant theory and causality to develop a unified theoretical framework that clarifies the differences between positional node embeddings and structural graph representations and emphasizes their correspondence.More specifically, (a) I show that structural representations and positional node embeddings have the same relationship as distributions and their samples; (b) I prove that all tasks that can be performed by positional node embeddings can also be performed by structural representations and vice-versa. Moreover, (c) I introduce new guidelines to creating and using positional node embeddings, which I hope will replace the less-than-optimal standard operating procedures used today. Finally, (d) I show that the concepts of *transductive* and *inductive* learning —commonly used to describe relational methods— are unrelated to the concepts of positional node embeddings and structural representations. This work was published as “On the equivalence between positional node embeddings and structural graph representations” Srinivasan and Ribeiro 2020 [11].

2. In Chapter 3, I propose a hypergraph neural network which exploits the incidence structure and hence works on real world sparse hypergraphs which have hyperedges of different cardinalities. Secondly, I provide a framework to learn provably expressive representations of the hypergraph which preserves properties of hypergraph and hyperedge isomorphism. Additionally, I introduce a new task on hypergraphs — namely

the variable sized hyperedge expansion (where given a partially observed hyperedge i.e. not all of its constituent vertices are known, the goal is to find nodes from the rest of the hypergraph which complete it — e.g. completing a partially known recipe) and also perform variable sized hyperedge classification. Finally, I demonstrate improved performance over existing baselines on majority of the hypergraph datasets using my proposed model. This work was published as “Learning over Families of Sets - Hypergraph Representation Learning for Higher Order Tasks” Srinivasan, Zheng, and Karypis 2021 [12].

3. In Chapter 4, I (a) introduce conditional invariances and provide a few guiding principles of conditional invariant representations as inductive biases, which serves as a generalization of group invariant neural networks (b) leverage conditional invariances to provide a framework to learn conformer invariant representations of proteins (c) propose a principled strategy to sample conformations for any given protein from the support of its protein conformer distribution (which consists of all its viable protein conformations) which captures their true flexibility [13], [14], while adhering to domain specific constraints such as those on dihedral angles, steric repulsions, among others (d) develop a Markov chain Monte Carlo learning framework which guides the sampling of protein conformations in such a way – so as to provide theoretical guarantees that the representations obtained by all conformations of a protein are identical. Finally, I demonstrate the empirical advantages of the proposed approach, wherein endowing baseline models with my proposed strategy shows noticeable improvements on four different tasks on proteins. This work is under submission as “Conditional Invariances for Conformer Invariant Protein Representations” Srinivasan, Ioannidis, Adeshina, Kakodkar, Karypis, and Ribeiro 2022 [15].

2. EQUIVALENCE BETWEEN STRUCTURAL GRAPH REPRESENTATIONS AND POSITIONAL NODE EMBEDDINGS

The theory of *structural* graph representations is a recently emerging field. It creates a link between relational learning and invariant theory. Interestingly, or rather unfortunately, there is no unified theory connecting node embeddings — low-rank matrix approximations, factor analysis, latent semantic analysis, etc.— with structural graph representations. Instead, conflicting interpretations have manifested over the last few years, that further confound practitioners and researchers alike.

For instance, consider the direction, *word embeddings* $\rightarrow$ *structural representations*, where the structural equivalence between *men* $\rightarrow$ *king* and *women* $\rightarrow$ *queen* is described as being obtained by just adding or subtracting their node embeddings (positions in the embedding space) [16], [17]. *Hence, can all (positional) node embeddings provide structural relationships akin to word analogies?* In the opposite direction, *structural representations* $\rightarrow$ *node embeddings*, graph neural networks (GNNs) are often optimized to predict edges even though their structural node representations are provably incapable of performing the task. For instance, the node representations of the *lynx* and the *orca* in Figure 2.1 are indistinguishable due to an isomorphic equivalence between the nodes, making any edge prediction task that distinguishes the edges of *lynx* and *orca* a seemly futile exercise.

To illustrate this visually, I provide an example using the food web of Figure 2.1. I do this by showcasing the difference between structural representations (Figure 2.2) and positional node embeddings (Figure 2.3) obtained by graph neural networks and SVD, respectively, on the same graph.

Figure 2.2 shows a 2-dimensional ($\mathbb{R}^2$) *structural representation* of nodes obtained by a standard GNN (obtained by the 1-WL GNN of [9]), optimized to try to predict links without the addition of any random edges). Node colors represent the mapping of the learned structural representation in $\mathbb{R}^2$ into the red and blue interval of RGB space $[0, 255]^2$. Note that isomorphic nodes are forcibly mapped into the same structural representations (have the same color), even though the representation is trained to predict links. Specifically, the

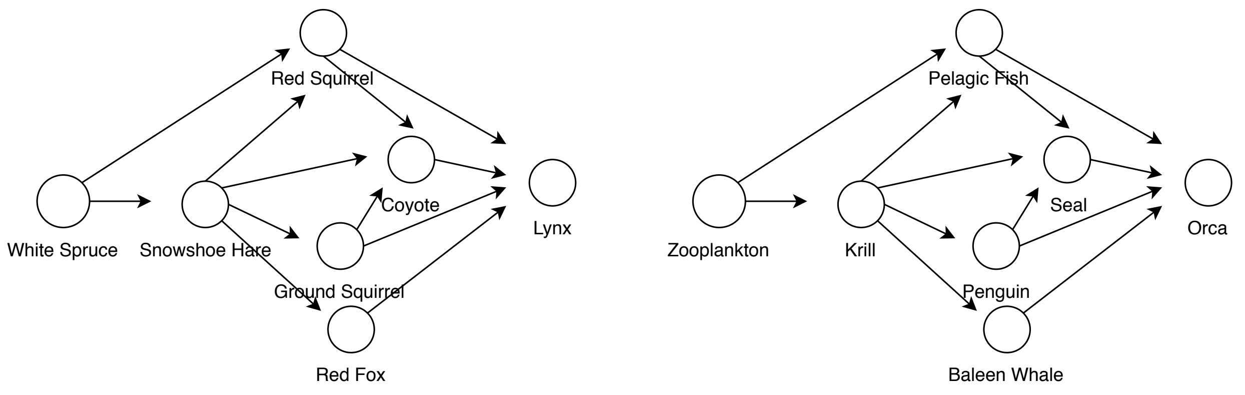


Figure 2.1. A food web example showing two disconnected components - the boreal forest [18] and the antarctic fauna [19]. The positional node embedding of the lynx and the orca can be different while their structural representation must be the same (due to the isomorphism).

lynx and the orca in Figure 2.2 get the same color since they are isomorphic, while the lynx and the coyote have different structural representations (different colors). The lynx, like the orca, is a top predator in the food web while the coyote is not a top predator. Hence, it is quite evident why GNN's are not traditionally used for link prediction: as the structural node representation is a color, using these representations is akin to asking "is a light pink node in Figure 2.2 connected to a light blue node?" We cannot answer this question unless we also specify which light pink (i.e. coyote or seal) and which light blue (i.e. orca or lynx) we are talking about. Hence, the failure to predict links.

Positional node embeddings, on the other hand, are often seen as a lower-dimensional projection of the rows and columns of the adjacency matrix $\boldsymbol{A}$ from $\mathbb{R}^n$ to $\mathbb{R}^d$, $d < n$, that preserves relative positions of the nodes in a graph. In Figure 2.3 I show the same food web graph, where now node colors map the values of the two leading (SVD) eigenvectors (of the undirected food web graph) into the $[0, 255]$ interval of blue and red intensity of RGB colors, respectively. The graph is made undirected, otherwise the left and right eigenvectors will be different and harder to represent visually. SVD is used here as a representative of positional node embedding methods. Note the lynx and the coyote now have close positional node embeddings (represented by colors which are closer in the color spectrum), while the positional node embeddings of the lynx and the orca are significantly different. Node

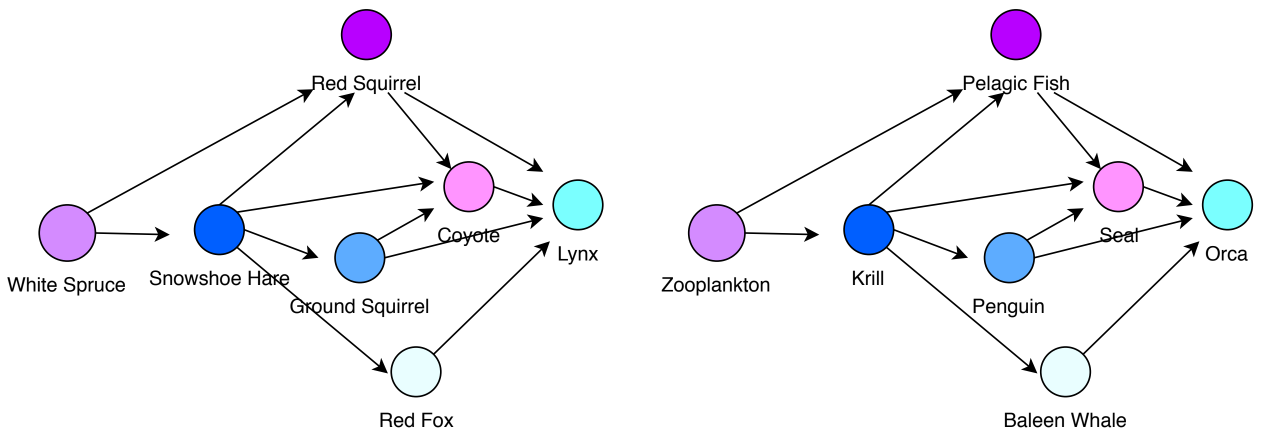


Figure 2.2. (Best in color) Food web graph of Figure 2.1 with node colors that map a two dimensional representation from a 1-WL GNN (GIN - [9]) into the $[0, 255]$ interval of blue and red intensity of RGB colors respectively. GIN is used a representative of structural representation of nodes. Structurally isomorphic nodes, obtain the same representation and are hence end up being visualised with the same color. Consequently, it is clear why structural node representations are used for node and graph classification, but not for link prediction.

embeddings are often seen as encoding the fact that the lynx and the coyote are part of a tightly-knit community, while the lynx and orca belong to distinct communities.

It is evident from Figure 2.3 why positional node embeddings are not traditionally used to predict node classes: predicting that the lynx, like the orca, is a top predator based on their node colors is difficult, since the coyote's color is very similar to that of the lynx, while the orca obtains a completely different color to the lynx. Relying on color shades (light and dark) for node classification is unreliable, since nodes with completely different structural positions in the food chain may have similar color shades.

On the other hand, link prediction using the colors in Figure 2.3 is rather trivial, since similar colors mean "closeness" in the graph. For instance, we may easily predict that the baleen whale also eats zooplankton.

Hence, this leads us to the question, are structural representations in general — and GNNs in particular — fundamentally incapable of performing link (dyadic) and multi-ary (polyadic) predictions tasks? GNNs, however, can perform node classification tasks, which

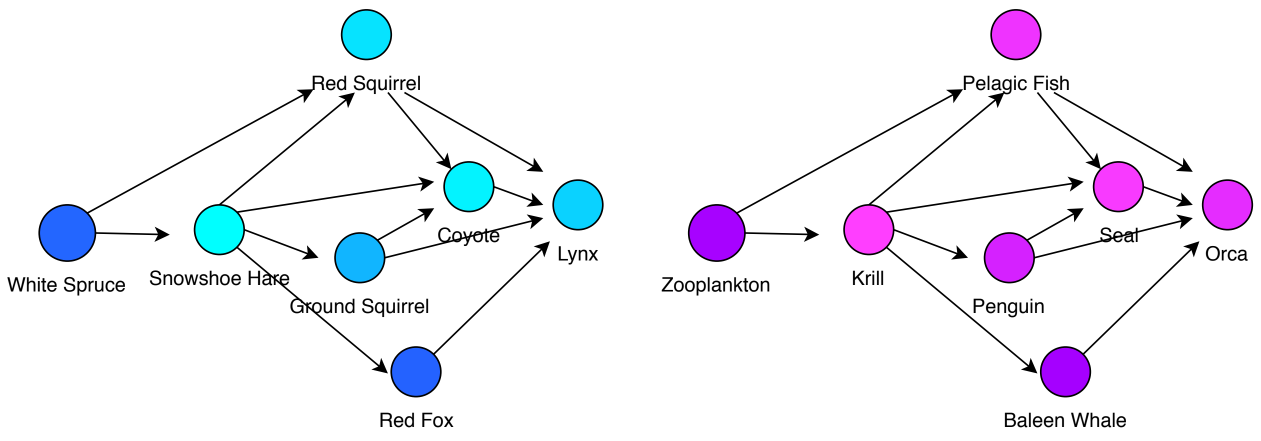


Figure 2.3. (Best in color) Food web graph of Figure 2.1 with node colors that map the values of the two leading (SVD) eigenvectors over the undirected graph into the $[0, 255]$ interval of blue and red intensity of RGB colors, respectively. SVD (run until convergence) is used as a representative of positional node embedding methods. The graph is made undirected, otherwise the left and right eigenvectors will be different and harder to represent visually. Note that nodes which are a part of the same connected component, obtain embeddings which are close in latent space, visually shown as similar colors. Consequently, it is clear that positional node embeddings can be used for link prediction and clustering.

is a task not associated with positional node embeddings. The goal of this chapter is to exactly address the above question (and more), and I start off with the contributions of this chapter.

Contributions: In this chapter I use invariant theory and axiomatic counterfactuals (causality) to develop a unified theoretical framework that clarifies the differences between node embeddings and structural representations and emphasizes their correspondence. More specifically, (a) I show that structural representations and node embeddings have the same relationship as distributions and their samples; (b) I prove that all tasks that can be performed by node embeddings can also be performed by structural representations and vice-versa. Moreover, (c) I introduce new guidelines to creating and using node embeddings, which we hope will replace the less-than-optimal standard operating procedures used today. Finally, (d) I show that the concepts of *transductive* and *inductive* learning —commonly used to describe relational methods— are unrelated to node embeddings and structural representations.

2.1 Preliminaries

This section introduces some basic definitions, attempting to keep the mathematical jargon in check, sometimes even sacrificing generality for clarity. I recommend [20] for a more formal description of some of the definitions in this section.

Definition 2.1.1 (Graph)**.** *Consider either a directed or an undirected attributed graph, denoted by* $G=(V,E,\boldsymbol{X},\mathbf{E})$*, where* V *is a set of* $n=|V|$ *vertices,* E *is the set of edges in* $V\times V$*, with matrix* $\boldsymbol{X}\in\mathbb{R}^{n\times k}, k>0$ *and 3-mode tensor* $\mathbf{E}\in\mathbb{R}^{n\times n\times k'}, k'>0$ *representing the node and edge features, respectively. The edge set has an associated adjacency matrix* $\boldsymbol{A}\in\{0,1\}^{n\times n}$*. In order to simplify notation, I compress* $\mathbf{E}$ *and* $\boldsymbol{A}$ *into a single tensor* $\mathbf{A}\in\mathbb{R}^{n\times n\times(k'+1)}$*. When explicit vertex and edge features and weights are unavailable, I consider* $\boldsymbol{X}=\mathbf{1}\mathbf{1}^T$ *and* $\mathbf{A}=\boldsymbol{A}$*, where* $\mathbf{1}$ *is a* $n\times 1$ *vector of ones. I will slightly abuse notation and denote the graph as* $G=(\mathbf{A},\boldsymbol{X})$*. Without loss of generality, I define number the nodes in* $V=\{1,\ldots,n\}$ *following the same ordering as the adjacency tensor* $\mathbf{A}$ *and the rows in* $\boldsymbol{X}$*. I denote* $\vec{S}$ *and a vector of the elements in* $S\in\mathcal{P}^{\star}(V)$ *sorted in ascending order, where* $\mathcal{P}^{\star}(V)$ *is the power set of* V *without the empty set.*

One of the most important operators in our mathematical toolkit will be that of a permutation action, orbits, $\mathcal{G}$-invariance, and $\mathcal{G}$-equivariance:

Definition 2.1.2 (Permutation action π)**.** *A permutation action* π *is a function that acts on any vector, matrix, or tensor defined over the nodes* V*, e.g.,* $(\boldsymbol{Z}_{\mathrm{i}})_{\mathrm{i}\in V}$*, and outputs an equivalent vector, matrix, or tensor with the order of the nodes permuted. I define* Π_n *as the set of all* $n!$ *such permutation actions.*

Definition 2.1.3 (Orbits)**.** *An orbit is the result of a group action* Π_n *acting on elements of a group correspond to bijective transformations of the space that preserve some structure of the space. The orbit of an element is the set of equivalent elements under action* Π_n*, i.e.,* $\Pi_n(x)=\{\pi(x)\mid\pi\in\Pi_n\}$.

Definition 2.1.4 ($\mathcal{G}$-equivariant and $\mathcal{G}$-invariant functions)**.** *Let Σ_n be the set of all possible attributed graphs G of size $n \geq 1$. More formally, Σ_n is the set of all tuples $(\mathbf{A}, \boldsymbol{X})$ with adjacency tensors $\mathbf{A}$ and corresponding node attributes $\boldsymbol{X}$ for n nodes. A function $g : \Sigma_n \to \mathbb{R}^{n\times\cdot}$ is $\mathcal{G}$-equivariant w.r.t. valid permutations of the nodes V, whenever any permutation action $\pi \in \Pi_n$ in the Σ_n space associated with the same permutation action of the nodes in the $\mathbb{R}^{n\times\cdot}$ space. A function $g : \Sigma_n \to \mathbb{R}^{\cdot}$ is $\mathcal{G}$-invariant whenever it is invariant to any permutation action $\pi \in \Pi_n$ in Σ_n.*

Definition 2.1.5 (Graph orbits & graph isomorphism)**.** *Let $G = (\mathbf{A}, \boldsymbol{X})$ be a graph with n nodes, and let $\Pi_n(G) = \{(\mathbf{A}', \boldsymbol{X}') : (\mathbf{A}', \boldsymbol{X}') = (\pi(\mathbf{A}), \pi(\boldsymbol{X})), \forall \pi \in \Pi_n\}$ be the set of all equivalent (isomorphic) graphs under the permutation action π. Two graphs $G_1 = (\mathbf{A}_1, \boldsymbol{X}_1)$ and $G_2 = (\mathbf{A}_2, \boldsymbol{X}_2)$ are said isomorphic iff $\Pi_n(G_1) = \Pi_n(G_2)$.*

Definition 2.1.6 (Node orbits & node isomorphism)**.** *The equivalence classes of the vertices of a graph G under the action of automorphisms are called vertex orbits. If two nodes are in the same node orbit, we say that they are isomorphic.*

In Figure 2.1, the lynx and the orca are isomorphic (they have the same node orbits). I now generalize Definition 2.1.6 to subsets of nodes $S \in \mathcal{P}^\star(V)$, where $\mathcal{P}^\star(V)$ is the power set of V without the empty set.

Definition 2.1.7 (Vertex subset orbits and joint isomorphism)**.** *The equivalence classes of k-sized subsets of vertices $S \in \mathcal{P}^\star(V)$ of a graph G under the action of automorphisms between the subsets are called vertex subset orbits, $k \geq 2$. If two proper subsets $S_1, S_2 \in \mathcal{P}^\star(V) \backslash V$ are in the same vertex subset orbit, we say they are jointly isomorphic.*

Next I define the relationship between structural representations and node embeddings.

2.2 A Unifying Theoretical Framework of Node Embeddings and Structural Representations

How are node embeddings and structural representations related? This section starts with a familiar, albeit naïve, view of the differences between node embeddings and structural representations, preparing the groundwork to later broadening and rectifying these into precise model-free mathematical statements using invariant theory. This broadening is needed since model-free node embeddings need not be related to *node closeness* in the graph (or to lower dimensional projections for that matter), as it is impossible to have a model-free definition of *closeness*.

A familiar interpretation of node embeddings: Node embeddings are often seen as a lower-dimensional projection of the rows and columns of the adjacency matrix $\boldsymbol{A}$ from $\mathbb{R}^n$ to $\mathbb{R}^d$, $d < n$, that preserves relative positions of the nodes in a graph [21], [22]; for instance, in Figure 2.1, the lynx and the coyote would have close node embeddings, while the node embeddings of the lynx and the orca would be significantly different. Node embeddings are often seen as encoding the fact that the lynx and the coyote are part of a tightly-knit community, while the lynx and orca belong to distinct communities. The *structural representation* of a node, on the other hand, shows which nodes have similar roles (structural similarities) on a graph; for instance, the lynx and the orca in Figure 2.1 must have the same structural representation, while the lynx and the coyote likely have different structural representations. The lynx, like the orca, is a top predator in the food web while the coyote is not a top predator.

The incompatibility of the familiar interpretation with the theory of structural graph representations: The above interpretation of node embeddings must be tied to a model that defines *closeness*. Structural graph representations are model-free. Hence, we need a model-free definition of node embedding to connect it with structural representations. Unfortunately, one cannot define *closeness* without a model. Hence, in the remainder of this paper, I abandon this familiar interpretation in favor of a model-free definition.

Roadmap: In what follows, I restate some existing model-free definitions of *structural graph representation* and introduce some new ones. Then, I introduce a model-free definition of node embeddings. I will retain the terminology *node embedding* for historical reasons, even though our *node embedding* need not be an embedding (a projection into lower dimensional space).

2.2.1 On Structural Representations

In what follows I use the terms link and edge interchangeably. Proofs are left to the Appendix.

Definition 2.2.1 (Structural node representations)**.** *The structural representation of node $v \in V$ in a graph $G = (\mathbf{A}, \boldsymbol{X})$ is the $\mathcal{G}$-invariant representation $\Gamma(v, \mathbf{A}, \boldsymbol{X})$, where $\Gamma : V \times \Sigma_n \to \mathbb{R}^d$, $d \geq 1$, such that $\forall u \in V$, $\Gamma(u, \mathbf{A}, \boldsymbol{X}) = \Gamma(\pi(u), \pi(\mathbf{A}), \pi(\boldsymbol{X}))$ for all permutation actions $\forall \pi \in \Pi_n$. Moreover, for any two isomorphic nodes $u, v \in V$, $\Gamma(u, \mathbf{A}, \boldsymbol{X}) = \Gamma(v, \mathbf{A}, \boldsymbol{X})$.*

Definition 2.2.2 (Most-expressive structural node representations $\Gamma^\star$)**.** *A structural representation of a node $v \in V$, $\Gamma^\star(v, \mathbf{A}, \boldsymbol{X})$, is most-expressive iff, $\forall u \in V$, there exists a bijective measurable map between $\Gamma^\star(u, \mathbf{A}, \boldsymbol{X})$ and the orbit of node u in $G = (\mathbf{A}, \boldsymbol{X})$ (Definition 2.1.7).*

Trivially, by Definitions 2.1.6 and 2.2.2, two graphs $G_1 = (\mathbf{A}_1, \boldsymbol{X}_1)$ and $G_2 = (\mathbf{A}_2, \boldsymbol{X}_2)$ are isomorphic (Definition 2.1.5) iff the most-expressive structural node representations $(\Gamma^\star(u, \mathbf{A}_1, \boldsymbol{X}_1))_{u \in V}$ and $(\Gamma^\star(v, \mathbf{A}_2, \boldsymbol{X}_2))_{v \in V}$ are the same up to a valid permutation $\pi \in \Pi_n$ of the nodes. In what follows $\mathcal{P}^\star$ is the power set excluding the empty set.

I now describe the relationship between structural node representations and node isomorphism.

Lemma 1. *Two nodes $v, u \in V$, have the same most-expressive structural representations $\Gamma^\star(v, \mathbf{A}, \boldsymbol{X}) = \Gamma^\star(u, \mathbf{A}, \boldsymbol{X})$ iff u and v are isomorphic nodes in $G = (\mathbf{A}, \boldsymbol{X})$.*

Having described representation of nodes, I now generalize these representations to subsets of V.

Definition 2.2.3 (Joint structural representation Γ)**.** *A joint structural representation of a graph with node set V is defined as $\Gamma : \mathcal{P}^\star(V) \times \Sigma_n \to \mathbb{R}^d$, $d \geq 1$. Furthermore, Γ is $\mathcal{G}$-invariant over all node subsets, i.e., $\forall S \in \mathcal{P}^\star(V)$ and $\forall (\mathbf{A}, \boldsymbol{X}) \in \Sigma_n$, it must be that $\Gamma(\vec{S}, \mathbf{A}, \boldsymbol{X}) = \Gamma(\pi(\vec{S}), \pi(\mathbf{A}), \pi(\boldsymbol{X}))$ for all permutation actions $\forall \pi \in \Pi_n$. Moreover, for any two isomorphic subsets $S, S' \in \mathcal{P}^\star(V)$, $\Gamma(\vec{S}, \mathbf{A}, \boldsymbol{X}) = \Gamma(\vec{S}', \mathbf{A}, \boldsymbol{X})$.*

I now mirror Definition 2.2.2 in our generalization of Γ:

Definition 2.2.4 (Most-expressive joint structural representations $\Gamma^\star$)**.** *A structural representation $\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X})$ of a non-empty subset $S \in \mathcal{P}^\star(V)$, of a graph $(\mathbf{A}, \boldsymbol{X}) \in \Sigma_n$, is most-expressive iff, there exists a bijective measurable map between $\Gamma^\star(\vec{U}, \mathbf{A}, \boldsymbol{X})$ and the orbit of U in G (Definition 2.1.7), $\forall U \in \mathcal{P}^\star(V)$ and $\forall(\mathbf{A}, \boldsymbol{X}) \in \Sigma_n$.*

Note, however, the failure to represent the link (*lynx*, *coyote*) in Figure 2.1 using the most-expressive node representations of the *lynx* and the *coyote*. A link needs to be represented by a joint representation of two nodes. For instance, we can easily verify from Definition 2.2.4 that $\Gamma^\star((\textit{lynx,coyote}), \mathbf{A}, \boldsymbol{X}) \neq \Gamma^\star((\textit{orca,coyote}), \mathbf{A}, \boldsymbol{X})$, even though $\Gamma^\star(\textit{lynx}, \mathbf{A}, \boldsymbol{X}) = \Gamma^\star(\textit{orca}, \mathbf{A}, \boldsymbol{X})$.

Next I show that joint prediction tasks only require joint structural representations. But first we need to show that any causal model defined through axiomatic counterfactuals [23] can be equivalently defined through noise outsourcing [24], a straightforward result that I was unable to find in the literature.

Lemma 2 (Causal modeling through noise outsourcing)**.** *Definition 1 of [23] gives a causal model as a triplet*

$$M = \langle U, V', F \rangle,$$

where U is a set of exogenous variables, V' is a set of endogenous variables, and F is a set of functions, such that in the causal model, $v'_{\mathrm{i}} = f(\vec{pa}_{\mathrm{i}}, u)$ is the realization of random variable $V'_{\mathrm{i}} \in V'$ and a sequence of random variables $\vec{PA}_{\mathrm{i}}$ with $PA_{\mathrm{i}} \subseteq V \backslash V'_{\mathrm{i}}$ as the endogenous variable parents of variable V'_{i} as given by a directed acyclic graph. Then, there exists a pure random noise ϵ and a set of (measurable) functions $\{g_u\}_{u \in U}$ such that for $V_{\mathrm{i}} \in V'$, V'_{i} can be equivalently defined as $v'_{\mathrm{i}} \overset{a.s.}{=} f(\vec{pa}_{\mathrm{i}}, g_u(\epsilon_u))$, where ϵ_u has joint distribution $(\epsilon_u)_{\forall u \in U} \overset{a.s.}{=} g'(\epsilon)$ for some Borel measurable function g' and a random variable $\epsilon \sim \text{Uniform}(0, 1)$. The latter defines M via noise outsourcing [24].

The proof of Lemma 2 is given in the Appendix. Lemma 2 defines the causal model entirely via endogenous variables, deterministic functions, and a pure random noise random variable.

We are now ready for our theorem showing that joint prediction tasks only require (most-expressive) joint structural representations.

Theorem 2.2.1. *Let $\mathcal{S} \subseteq \mathcal{P}^\star(V)$ be a set of non-empty subsets of the vertices V. Let $\boldsymbol{Y}(\mathcal{S}, \mathbf{A}, \boldsymbol{X}) = (Y(\vec{S}, \mathbf{A}, \boldsymbol{X}))_{S \in \mathcal{S}}$ be a sequence of random variables defined over the sets $S \in \mathcal{S}$ of a graph $G = (\mathbf{A}, \boldsymbol{X})$, that are invariant to the ordering of $\vec{S}, S \in \mathcal{S}$, such that $Y(\vec{S_1}, \mathbf{A}, \boldsymbol{X}) \overset{d}{=} Y(\vec{S_2}, \mathbf{A}, \boldsymbol{X})$ for any two jointly isomorphic subsets $S_1, S_2 \in \mathcal{S}$ (Definition 2.1.7), where $\overset{d}{=}$ means equality in their marginal distributions. Then, there exists a measurable function φ such that, $\boldsymbol{Y}(\mathcal{S}, \mathbf{A}, \boldsymbol{X}) \overset{a.s.}{=} (\varphi(\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X}), \epsilon_S))_{S \in \mathcal{S}}$, where ϵ_S is the random noise that defines the exogenous variables of Lemma 2, with joint distribution $p((\epsilon_{S'})_{\forall S' \in \mathcal{S}})$ independent of $\mathbf{A}$ and $\boldsymbol{X}$.*

Theorem 2.2.1 extends Theorem 12 of [20] in multiple ways: (a) to all subsets of nodes, $S \in \mathcal{P}^\star(V)$, (b) to include causal language, and, most importantly, (c) to showing that any prediction task that can be defined over S, requires only a most-expressive joint structural representation over S. For instance, any task with $|S| = 2$ predicting a missing link (u, v) on a graph $G = (\mathbf{A}, \boldsymbol{X})$, requires only the most-expressive structural representation $\Gamma^\star((u, v), \mathbf{A}, \boldsymbol{X})$. Note that, in order to predict directed edges, we must use $\Gamma^\star((u, v), \mathbf{A}, \boldsymbol{X})$ to also predict the edge's direction: $\rightarrow$, $\leftarrow$, $\leftrightarrow$, but a detailed procedure showing how to predict directed edges is relegated to a future journal version of this paper. Theorem 2.2.1 also includes node tasks for $|S| = 1$, hyperedge tasks for $2 < |S| < n$, and graph-wide tasks for $S = V$.

Remark 1 (GNNs and link prediction)**.** *Even though structural node representations of GNNs are not able to predict edges, GNNs are often still optimized to predict edges (e.g., [7], [9]) in transfer learning tasks. This optimization objective guarantees that any small topological differences between two nearly-isomorphic nodes without an edge will be amplified, while differences between nodes with an edge will be minimized. Hence, the topological differences in a close-knit community will be minimized in the representation. This procedure is an interesting way to introduce homophily in structural representations and should work well for node classification tasks in homophilic networks (where node classes tend to be clustered).*

We now turn our attention to node embeddings and their relationship with joint representations.

2.2.2 On (Positional) Node Embeddings

Definition 2.2.5 (Node Embeddings)**.** *The node embeddings of a graph* $G = (\mathbf{A}, \boldsymbol{X})$ *are defined as joint samples of random variables* $(\boldsymbol{Z}_\mathrm{i})_{\mathrm{i}\in V}|\mathbf{A}, \boldsymbol{X} \sim p(\cdot|\mathbf{A}, \boldsymbol{X})$, $\boldsymbol{Z}_\mathrm{i} \in \mathbb{R}^d$, $d \geq 1$, *where* $p(\cdot|\mathbf{A}, \boldsymbol{X})$ *is a* $\mathcal{G}$*-equivariant probability distribution on* $\mathbf{A}$ *and* $\boldsymbol{X}$*, that is,* $\pi(p(\cdot|\mathbf{A}, \boldsymbol{X})) = p(\cdot|\pi(\mathbf{A}), \pi(\boldsymbol{X}))$ *for any permutation* $\pi \in \Pi_n$*.*

Essentially, Definition 2.2.5 says that the probability distribution $p(\boldsymbol{Z}|\mathbf{A}, \boldsymbol{X})$ of a node embedding $\boldsymbol{Z}$ must be $\mathcal{G}$-equivariant on $\mathbf{A}$ and $\boldsymbol{X}$. This is the only property we require to define a node embedding. Next, I show that the node embeddings given by Definition 2.2.5 cover a wide range of embedding methods in the literature.

Corollary 1. *The node embeddings in Definition 2.2.5 encompass embeddings given by matrix and tensor factorization methods —such as Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), implicit matrix factorization (a.k.a. word2vec)–, latent embeddings given by Bayesian graph models —such as Probabilistic Matrix Factorizations (PMFs) and variants—, variational autoencoder methods and graph neural networks that use random lighthouses to extract node embedddings.*

The proof of Corollary 1 is in the Appendix, along with the references of each of the methods mentioned in the corollary. The output of some of the methods described in Corollary 1 is deterministic, and for those, the probability density $p(\boldsymbol{Z}|\mathbf{A}, \boldsymbol{X})$ is a Dirac delta. In practice, however, even deterministic methods use algorithms whose outputs depend on randomized initial conditions, which will also satisfy Corollary 1.

I now show that permutation equivariance implies two isomorphic nodes (or two subsets of nodes) must have the same marginal distributions over $\boldsymbol{Z}$:

Lemma 3. *The permutation equivariance of* p *in Definition 2.2.5 implies that, if two proper subsets of nodes* $S_1, S_2 \in \mathcal{P}^\star(V)\backslash V$ *are isomorphic, then their marginal node embedding distributions must be the same up to a permutation, i.e.,* $p((\boldsymbol{Z}_\mathrm{i})_{\mathrm{i}\in S_1}|\mathbf{A}, \boldsymbol{X}) = \pi(p((\boldsymbol{Z}_\mathrm{j})_{\mathrm{j}\in S_2}|\mathbf{A}, \boldsymbol{X}))$ *for some appropriate permutation* $\pi \in \Pi_n$*.*

Hence, the critical difference between the structural node representation vector $(\Gamma(v, \mathbf{A}, \boldsymbol{X}))_{v\in V}$ in Definition 2.2.1 and node embeddings $\boldsymbol{Z}$ in Definition 2.2.5, is that the vector $(\Gamma(v, \mathbf{A}, \boldsymbol{X}))_{v\in V}$

must be $\mathcal{G}$-equivariant while $\boldsymbol{Z}$ *need not be* —even though $\boldsymbol{Z}$'s distribution must be $\mathcal{G}$-equivariant. This seemly trivial difference has tremendous consequences, which I explore in the reminder of this section.

Next, I show that node embeddings $\boldsymbol{Z}$ cannot have any extra information about G that is not already contained in a most-expressive structural representation $\Gamma^\star$.

Theorem 2.2.2 (The statistical equivalence between node embeddings and structural representations)**.** *Let* $\boldsymbol{Y}(\mathcal{S}, \mathbf{A}, \boldsymbol{X}) = (Y(\vec{S}, \mathbf{A}, \boldsymbol{X}))_{S\in\mathcal{S}}$ *be as in Theorem 2.2.1. Consider a graph* $G = (\mathbf{A}, \boldsymbol{X}) \in \Sigma_n$*,* $n \geq 2$*. Let* $\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X})$ *be a most-expressive structural representation of nodes* $S \in \mathcal{P}^\star(V)$ *in* G*. Then,*

$$Y(\vec{S}, \mathbf{A}, \boldsymbol{X}) \perp\!\!\!\perp_{\Gamma^\star(\vec{S},\mathbf{A},\boldsymbol{X})} \boldsymbol{Z}|\mathbf{A}, \boldsymbol{X}, \quad \forall S \in \mathcal{S},$$

for any node embedding matrix $\boldsymbol{Z}$ *that satisfies Definition 2.2.5, where* $A \perp\!\!\!\perp_B C$ *means* A *is independent of* C *given* B*. Finally,* $\forall(\mathbf{A}, \boldsymbol{X}) \in \Sigma_n$*, there exists a most-expressive node embedding* $\boldsymbol{Z}^\star|\mathbf{A}, \boldsymbol{X}$ *such that,*

$$\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X}) = \mathbb{E}_{\boldsymbol{Z}^\star}[f^{(|S|)}((\boldsymbol{Z}^\star_v)_{v\in S})|\mathbf{A}, \boldsymbol{X}], \quad \forall S \in \mathcal{S},$$

for some appropriate collection of functions $\{f^{(k)}(\cdot)\}_{k=1,\ldots,n}$*.*

The proof of Theorem 2.2.2 is given in the Appendix. Note that the most-expressive embedding $\boldsymbol{Z}^\star|\mathbf{A}, \boldsymbol{X}$ extends the insight used to make GNNs more expressive in [25] to a more general procedure.

Theorem 2.2.2 implies that, *for any graph prediction task, node embeddings carry no information beyond that of structural representations.* A less attentive reader may think this creates an apparent paradox, since one cannot predict a property $Y((\textit{lynx,coyote}), \mathbf{A}_{\text{food web}}, \boldsymbol{X}_{\text{food web}})$ in Figure 2.1 from structural node embeddings, since $\Gamma(\textit{lynx}, \mathbf{A}, \boldsymbol{X}) = \Gamma(\textit{orca}, \mathbf{A}, \boldsymbol{X})$. The resolution of the paradox is to note that Theorem 2.2.2 describes the prediction of a link through a pairwise structural representation $\Gamma((\textit{lynx,coyote}), \mathbf{A}_{\text{food web}}, \boldsymbol{X}_{\text{food web}})$, and we may not be able to do the same task with structural node representations alone. An interesting question for future work is how well can we learn distributions (representations) from (node embeddings) samples, extending [26] to graph representations.

Other equally important consequences of Theorem 2.2.2 are: (a) any sampling approach obtaining node embeddings $\boldsymbol{Z}$ is valid as long as the distribution is $\mathcal{G}$-equivariant (Definition 2.2.5), noting that isomorphic nodes must have the same marginal distributions (per Lemma 3). (b) Interestingly, convex optimization methods for matrix factorization can be seen as variance-reduction techniques with no intrinsic value beyond reducing variance. (c) Methods that give unique node embeddings —if the embedding of any two isomorphic nodes are different— are provably incorrect when used to predict graph relationships since they are permutation-sensitive.

Remark 2 (Some GNN methods give node embeddings not structural representations)**.** *The random edges added by GraphSAGE [7] and GIN [9] random walks make these methods node embeddings rather than structural node representations, according to Definition 2.2.5. To transform them back to structural node representations, one must average over all such random walks.*

The following corollaries describe other consequences of Theorem 2.2.2:

Corollary 2. *The link prediction task between any two nodes $u, v \in V$ depends only on the most-expressive tuple representation $\Gamma^\star((u,v), \mathbf{A}, \mathbf{X})$. Moreover, $\Gamma^\star((u,v), \mathbf{A}, \mathbf{X})$ always exists for any graph $(\mathbf{A}, \boldsymbol{X})$ and nodes (u,v). Finally, given most-expressive node embeddings $\boldsymbol{Z}^\star$, there exists a function f such that $\Gamma^\star((u,v), \mathbf{A}, \mathbf{X}) = \mathbb{E}_{\boldsymbol{Z}^\star}[f(\boldsymbol{Z}^\star_u, \boldsymbol{Z}^\star_v)]$, $\forall u, v$.*

A generalization of Corollary 2 is also possible, where Theorem 2.2.2 is used to allow us to create joint representations from simpler node embedding sampling methods.

Corollary 3. *Sample $\boldsymbol{Z}$ according to Definition 2.2.5. Then, we can learn a k-node structural representation of a subset of k nodes $S \in \mathcal{P}^\star(V)$, $|S| = k$, simply by learning a function $f^{(k)}$ whose average $\Gamma(\vec{S}, \mathbf{A}, \boldsymbol{X}) = \mathbb{E}[f^{(k)}((Z_v)_{v \in S})]$ can be used to predict $Y(\vec{S}, \mathbf{A}, \boldsymbol{X})$.*

The proof of Corollary 3 is in the Appendix. Finally, I show that the concepts of transductive and inductive learning are unrelated to the notions of node embeddings and structural representations.

Corollary 4. *Transductive and inductive learning are unrelated to the concepts of node embeddings and structural representations.*

Corollary 4 clears a confusion that, I believe, arises because traditional applications of node embeddings use a single Monte Carlo sample of $\boldsymbol{Z}|\mathbf{A}, \boldsymbol{X}$ to produce a structural representation (e.g., [17]). Inherently, a classifier learned with such a poor structural representation may fail to generalize over the test data, and will be deemed *transductive*.

Corollary 5. *Merging a node embeddings sampling scheme with GNNs can increase the structural representation power of GNNs.*

Corollary 5 is a direct consequence of Theorem 2.2.2, with [25] showing RP-GNN as a concrete method to do so.

2.3 Results

This section focuses on applying the lessons learned in Section 2.2 in four tasks, divided into two common goals. The goal of the first three tasks is to show that, as described in Theorem 2.2.2, node embeddings can be used to create expressive structural embeddings of nodes, tuples, and triads. These representations are then subsequently used to make predictions on downstream tasks with varied node set sizes. The tasks also showcase the added value of using multiple node embeddings (Monte Carlo) samples to estimate structural representations, both during training and testing. Moreover, showcasing Theorem 2.2.1 and the inability of node representations to capture joint structural representations, these tasks show that structural node representations are useless in prediction tasks over more than one node, such as links and triads. The goal of fourth task is to showcase how multiple Monte Carlo samples of node embeddings are required to observe the fundamental relationship between structural representations and node embeddings predicted by Theorem 2.2.2.

An important note: The proposed theoretical framework is not limited to the way I generate node embeddings. For example, the theoretical framework can use SVD in an inductive setting, where I train a classifier in one graph and test in a different graph, which was thought not possible previously with SVD. SVD with our theoretical framework is denoted MC-SVD, to emphasize the importance of Monte Carlo sampling in building better structural representations. Alternatively, more expressive node embeddings can be obtained using Colliding Graph Neural Networks (CGNN), as I show in the Appendix (Section A.3 and Section A.4)

2.3.1 Quantitative Results

In what follows, I evaluate structural representations estimated from four node embedding techniques, namely GIN [9], RP-GIN [25], 1-2-3 GNN [10], MC-SVD and CGNN. I classify GIN, RP-GIN and 1-2-3 GNN as node embedding techniques, as they employ the unsupervised learning procedure of [7]. These were chosen because of their potential extra link and triad representation power over traditional structural representation GNNs. All node embedding methods are evaluated by their effect in estimating good structural representation for downstream task accuracy. I partition $G = (\mathbf{A}, \boldsymbol{X})$ into three non-overlapping induced subgraphs, namely $G_{\text{train}} = (\mathbf{A}_{\text{train}}, \boldsymbol{X}_{\text{train}})$, $G_{\text{val}} = (\mathbf{A}_{\text{val}}, \boldsymbol{X}_{\text{val}})$ and $G_{\text{test}} = (\mathbf{A}_{\text{test}}, \boldsymbol{X}_{\text{test}})$, which I use for training, validation and testing, respectively. In learning all four node embedding techniques, I only make use of the graphs G_{train} and G_{val}. All the four models used here have never seen the test graph G_{test} before test time —i.e., all our node embedding methods, used in the framework of Theorem 2.2.2, behave like inductive methods.

Monte Carlo joint representations during an **unsupervised** *learning phase:* A key component of our optimization is learning joint representations from node embeddings —as per Theorem 2.2.2. For this, at each gradient step (in practice, I do at each epoch), I perform a Monte Carlo sample of the node embeddings $\boldsymbol{Z}|\mathbf{A}, \boldsymbol{X}$. This, procedure optimizes a proper upper bound on the empirical loss, if the loss is the negative log-likelihood, cross-entropy, or a square loss. The proof is trivial by Jensen's inequality. For GIN, RP-GIN and 1-2-3 GNN, I add random edges to the graph following a random walk at each epoch [7]. For the MC-SVD procedure, I use the left eigenvector matrix obtained by: (1) a random seed, (2) a random input permutation of the adjacency matrix, and (3) a single optimization step, rather than running SVD until it converges. I also present results with MC-SVD†, which is the same procedure as before, but runs SVD until convergence —noting that the latter is likely to give deterministic results in large real-world graphs.

Table 2.1. Micro F1 score on three distinct tasks averaged over 12 runs with standard deviation in parenthesis. The number within the parenthesis beside the model name indicates the number of Monte Carlo samples used in the estimation of the structural representation. MC-SVD†(1) denotes the SVD procedure run until convergence with one Monte Carlo sample for the representation. Bold values show maximum empirical average, and multiple bolds happen when its standard deviation overlaps with another average. Results for Citeseer are provided in the Appendix in Table A.1.

	Node Classification			Link Prediction			Triad Prediction		
	Cora	Pubmed	PPI	Cora	Pubmed	PPI	Cora	Pubmed	PPI
Random	0.143	0.333	0.5^{121}	0.500	0.500	0.500	0.250	0.250	0.250
GIN(1)	0.646(0.021)	**0.878(0.006)**	0.533(0.003)	0.526(0.029)	0.513(0.048)	0.604(0.018)	0.280(0.010)	0.430(0.019)	0.400(0.006)
GIN(5)	0.676(0.031)	**0.880(0.003)**	0.535(0.004)	0.491(0.019)	0.517(0.028)	0.609(0.012)	0.284(0.017)	0.422(0.024)	0.397(0.004)
GIN(20)	0.678(0.024)	**0.880(0.002)**	0.536(0.003)	0.514(0.026)	0.512(0.042)	0.603(0.010)	0.281(0.010)	0.422(0.028)	0.399(0.004)
RP-GIN(1)	0.655(0.023)	**0.879(0.002)**	0.534(0.005)	0.506(0.016)	0.616(0.048)	0.605(0.011)	0.283(0.013)	0.423(0.024)	0.400(0.005)
RP-GIN(5)	0.681(0.022)	**0.881(0.004)**	0.534(0.004)	0.498(0.016)	0.637(0.038)	0.612(0.006)	0.285(0.025)	0.429(0.024)	0.399(0.009)
RP-GIN(20)	0.675(0.032)	**0.879(0.005)**	0.533(0.003)	0.518(0.017)	0.619(0.032)	0.603(0.007)	0.279(0.011)	0.418(0.011)	0.393(0.003)
1-2-3 GNN(1)	0.319(0.017)	0.412(0.005)	0.403(0.003)	0.501(0.007)	0.495(0.018)	0.502(0.005)	0.280(0.010)	0.416(0.020)	0.250(0.003)
1-2-3 GNN(5)	0.321(0.008)	0.395(0.065)	0.405(0.001)	0.501(0.018)	0.500(0.002)	0.501(0.003)	0.285(0.015)	0.418(0.029)	0.251(0.005)
1-2-3 GNN(20)	0.324(0.010)	0.462(0.113)	0.401(0.007)	0.501(0.007)	0.499(0.002)	0.501(0.008)	0.285(0.014)	0.419(0.026)	0.254(0.008)
MC-SVD†(1)	0.665(0.014)	0.810(0.009)	0.523(0.005)	0.588(0.029)	0.807(0.024)	**0.755(0.010)**	0.336(0.038)	0.515(0.077)	0.532(0.010)
MC-SVD(1)	0.667(0.017)	0.825(0.007)	0.521(0.006)	0.583(0.020)	0.818(0.032)	**0.755(0.008)**	0.304(0.034)	0.518(0.065)	0.529(0.006)
MC-SVD(5)	0.669(0.013)	0.842(0.015)	0.556(0.009)	0.572(0.019)	**0.848(0.038)**	**0.754(0.006)**	0.306(0.037)	**0.567(0.061)**	**0.544(0.008)**
MC-SVD(20)	0.672(0.013)	0.855(0.010)	0.591(0.009)	0.580(0.021)	**0.868(0.029)**	**0.762(0.010)**	0.300(0.033)	**0.546(0.029)**	**0.550(0.007)**
CGNN(1)	0.468(0.026)	0.686(0.020)	0.545(0.010)	0.682(0.026)	0.587(0.027)	0.661(0.015)	0.352(0.028)	0.404(0.014)	0.414(0.009)
CGNN(5)	0.641(0.022)	0.808(0.008)	0.637(0.014)	**0.707(0.027)**	0.585(0.037)	0.704(0.012)	**0.414(0.045)**	0.417(0.018)	0.463(0.026)
CGNN(20)	**0.726(0.024)**	0.831(0.010)	**0.707(0.015)**	**0.712(0.041)**	0.581(0.039)	0.738(0.011)	**0.405(0.034)**	0.419(0.017)	0.498(0.021)

Monte Carlo joint representations during a **supervised** *learning phase:* During the supervised phase, I first estimate a structural joint representation $\hat{\Gamma}(\vec{S}, \mathbf{A}, \boldsymbol{X})$ as the average of $m \in \{1, 5, 20\}$ Monte Carlo samples of a permutation-invariant function [27], [28] (sum-pooling followed by an MLP) applied to a sampled node embedding $(\boldsymbol{Z}_v)_{v \in S} | \mathbf{A}, \boldsymbol{X}$. Then, using $\hat{\Gamma}(\vec{S}, \mathbf{A}, \boldsymbol{X})$, I predict the corresponding target variable $Y(S, \mathbf{A}, \boldsymbol{X})$ of each task using an MLP. The node sets of our tasks $S \subseteq V$, have sizes $|S| \in \{1, 2, 3\}$, corresponding to node classification, link prediction, and triad prediction tasks, respectively.

Datasets: I consider four graph datasets used by [7], namely Cora, Citeseer, Pubmed [29], [30] and PPI [31]. Cora, Citeseer and Pubmed are citation networks, where vertices represent papers, edges represent citations, and vertex features are bag-of-words representation of the document text. The PPI (protein-protein interaction) dataset is a collection of multiple graphs representing the human tissue, where vertices represent proteins, edges represent interactions across them, and node features include genetic and immunological information. Train, validation and test splits are used as proposed by [32] (see Table A.2 in the Appendix). Further dataset details can be found in the Appendix.

Node classification task: This task predicts node classes for each of the four datasets. In this task, structural node representations are enough. The structural node representation is used to classify nodes into different classes using an MLP, whose weights are trained in a supervised manner using the same splits as described above. In Cora, Citeseer and Pubmed, each vertex belongs only to a single class, whereas in the PPI graph dataset, nodes could belong to multiple classes.

Link prediction task: Here, I predict a small fraction of edges and non-edges in the test graph, as well as identify all false edges and non-edges (which were introduced as a corruption of the original graph) between different pairs of nodes in the graph. Specifically, I use joint tuple representations $\Gamma((u,v), \mathbf{A}, \boldsymbol{X})$, for $u, v \in V$, as prescribed by Theorem 2.2.2. Since, datasets are sparse in nature, and a trivial 'non-edges' predictor would result in a very high accuracy, we balance the train and validation and test splits to contain an equal number of edges and non-edges.

Triad prediction task: This task involves the prediction of triadic interaction as well as identification of possible fake interactions in the data between the three nodes under consideration. In this case, I use joint triadic representations $\Gamma((u,v,h), \mathbf{A}, \boldsymbol{X})$, for $u, v, h \in V$, as prescribed by Theorem 2.2.2. Here, I ensure that edge corruptions are dependent. I treat the graphs as being undirected in accordance with previous literature, and predict the number of true (uncorrupted) edges between the three nodes. Again, to handle the sparse nature of the graphs, I use a balanced dataset for train, validation, and test.

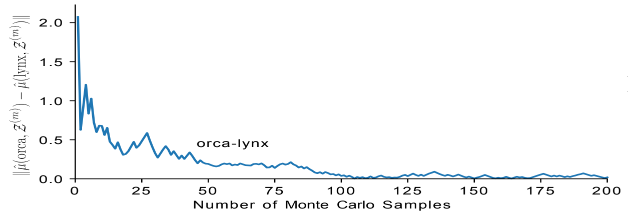
In Table 4.2 I present Micro-F1 scores for all four models over the three tasks. First, I note how more Monte Carlo samples at test time tend to increase test accuracy. In *node classification tasks*, we note that structural node representations from CGNN node embeddings significantly outperform other methods in two of the three datasets (the harder tasks). In *link prediction tasks*, the low accuracy of GNN-based methods (close to random) showcases the little extra-power of GIN and RP-GIN sampling schemes have over the the inability of structural node representations to predict links. Surprisingly, in *triads predictions*, the accuracy of GNN-based methods is much above random in some datasets, but still far from other node embedding methods. In *link and triad prediction tasks*, MC-SVD and CGNN share the lead with MC-SVD winning on Pubmed and PPI, and CGNN being significantly more accurate on Cora. Although, the 1-2-3 GNN is based on the two-Weisfeiler-Lehman (2-WL) algorithm (pairwise Weisfeiler-Lehman algorithm [33]), which provides tuple representations that can be exploited towards link prediction, it is however an approximation primarily designed for graph classification tasks. Unfortunately, the 1-2-3 GNN performs quite poorly on all our tasks (node classification, link and triad prediction), indicating the need for a task-specific approximation of 2-WL GNN's. Results for Citeseer, in the Appendix, show similar results.

2.3.2 Qualitative Results

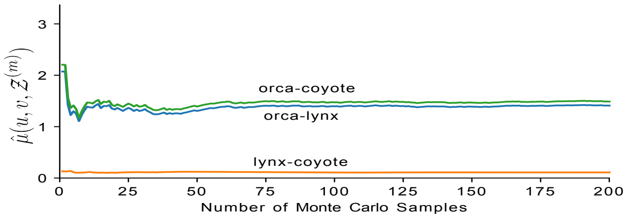
I now investigate the transformation of node embedding into node and link structural representations. Theorem 2.2.2 shows that the average of a function over node embedding Monte Carlo samples gives node and link embedding. In this experiment, I empirically test Theorem 2.2.2, by creating structural representations from the node embedding random matrix $\boldsymbol{Z}$, defined as the left eigenvector matrix obtained through SVD (ran until convergence), with the sources of randomness being due to a random permutation of the adjacency matrix given as input to the SVD method and the random seed it uses. Consider m such embedding matrices Monte Carlo samples, $\mathcal{Z}^{(m)} = \{\boldsymbol{Z}^{(\mathrm{i})}\}_{\mathrm{i}=1}^{m}$.

Structural node representations from node embeddings: According to Theorem 2.2.2, the average $\mathbb{E}[\boldsymbol{Z}_{v,\cdot}|\mathbf{A}]$ is a valid structural representation of node $v \in V$ in the adjacency matrix $\mathbf{A}$ of Figure 2.1. To test this empirically, I consider the unbiased estimator $\hat{\mu}(v, \mathcal{Z}^{(m)}) = \frac{1}{m}\sum_{\mathrm{i}=1}^{m} \boldsymbol{Z}_{v,\cdot}^{(\mathrm{i})}$, $v \in V$, where $\lim_{m\to\infty} \hat{\mu}(v, \mathcal{Z}^{(m)}) \overset{a.s.}{=} \mathbb{E}[\boldsymbol{Z}_{v,\cdot}|\mathbf{A}]$. Figure 2.4a shows the Euclidean distance between the empirical structural representations $\hat{\mu}(\mathrm{orca}, \mathcal{Z}^{(m)})$ and $\hat{\mu}(\mathrm{lynx}, \mathcal{Z}^{(m)})$ as a function of $m \in [1, 200]$. As expected, because these two nodes are isomorphic, $\|\hat{\mu}(\mathrm{orca}, \mathcal{Z}^{(m)}) - \hat{\mu}(\mathrm{lynx}, \mathcal{Z}^{(m)})\| \to 0$ as m grows, with $m = 100$ giving reasonably accurate results.

Structural link representations from node embeddings: According to Theorem 2.2.2, the average $\mathbb{E}[f^{(2)}(\boldsymbol{Z}_{u,\cdot}, \boldsymbol{Z}_{v,\cdot})|\mathbf{A}]$ of a function $f^{(2)}$ is a valid structural representation of a link with nodes $u, v \in V$ in the adjacency matrix $\mathbf{A}$ of Figure 2.1. As an example, I use $f^{(2)}(a, b) = \|a - b\|$, and define the unbiased estimator $\hat{\mu}(u, v, \mathcal{Z}^{(m)}) = \frac{1}{m}\sum_{\mathrm{i}=1}^{m} \|Z_u^{(\mathrm{i})} -$



(a) Difference in avg. structural node representations.



(b) Average structural link representations

Figure 2.4. Structural Representations for nodes and links using multiple samples obtained using MC-SVD on the disconnected food web graph shown in Figure 2.1.

$Z_v^{(i)}\|, \forall u, v \in V$, where $\lim_{m\to\infty} \hat{\mu}(u, v, \mathcal{Z}^{(m)}) \overset{a.s.}{=} \mathbb{E}[\|\boldsymbol{Z}_{u,\cdot} - \boldsymbol{Z}_{v,\cdot}\||\mathbf{A}]$. Figure 2.4b shows the impact of increasing the number of Monte Carlo samples m over the empirical structural representation of links. We observe that although the empirical node representations of the orca and the lynx seem to converge to the same value, $\lim_{m\to\infty} \hat{\mu}(\text{orca}, \mathcal{Z}^{(m)}) = \lim_{m\to\infty} \hat{\mu}(\text{lynx}, \mathcal{Z}^{(m)})$, their empirical joint representations with the coyote converge to different values, $\lim_{m\to\infty} \hat{\mu}(\text{lynx, coyote}, \mathcal{Z}^{(m)}) \neq \lim_{m\to\infty} \hat{\mu}(\text{orca, coyote}, \mathcal{Z}^{(m)})$, as predicted by Theorem 2.2.2. Also note a similar (but weaker) trend for $\lim_{m\to\infty} \hat{\mu}(\text{orca, lynx}, \mathcal{Z}^{(m)}) \neq \lim_{m\to\infty} \hat{\mu}(\text{orca, coyote}, \mathcal{Z}^{(m)})$, showing these three tuples to be structurally different.

2.4 Related Work

Node Embeddings vs Structural Representations: Prior works have categorized themselves as one of either node embedding methods or methods which learn structural representations. This artificial separation, consequently led to little contemplation of the relation between the two, restricting each of these approaches to a certain subsets of downstream tasks on graphs. Node embeddings were arguably first defined in 1904, through Spearman's common factors. Ever since, there has never been a universal definition of node embedding: node embeddings were simply the product of a particular method. This literature features a myriad of methods, e.g., matrix factorization [35]–[38], implicit matrix factorization [1], [3], [5], [17], [39], Bayesian factor models [4], [6], and some types of neural networks [40]–[43].

Arguably, the most common interpretation of node embeddings borrows from definitions of graph (node) embeddings in metric spaces: a measure of relative node *closeness* [21],

[22], [44]–[50]. Even in non-metric methods, such as word2vec [17] and Glove [51], the embeddings have properties similar to those of metric spaces [52]. Note that the definition of *close* varies from method to method, i.e., it is *model-dependent*. Still, this interpretation of *closeness* is the reason why their downstream tasks are often link prediction and clustering. However, once the literature started defining relative node closeness with respect to structural neighborhood similarities (e.g., [53]–[55]), node embeddings and structural representations became more strangely entangled.

Structural representations have an increasing body of literature focused on node and whole-graph classification tasks. Theoretically, these works abandon metric spaces in favor of a group-theoretic description of graphs [20], [25], [56]–[58], with connections to finite exchangeability and prior work on multilayer perceptrons [59]. Graph neural networks (GNNs) (e.g., [2], [7], [9], [60]–[62] among others) exploit this approach in tasks such as node and whole-graph classification. [10] proposes a higher-order Weisfeller-Lehman GNN (WL-k-GNN), which is shown to get better accuracy in graph classification tasks than traditional (WL-1) GNNs. Unfortunately, [10] focused only on graph-wide tasks, missing the fact that WL-2 GNN should be able to also perform link prediction tasks (Theorem 2.2.1), unlike WL-1 GNNs. More recently, graph neural networks have also been employed towards relational reasoning as well as matrix completion tasks [63]–[67]. However, these GNN's, in general, learn node embeddings rather than structural node representations, which are then exploited towards link prediction. GNN-like architectures have been used to simulate dynamic programming algorithms [68], which is unrelated to graphs and outside the scope of this work.

To the best of our knowledge, our work is the first to provide the theoretical foundations connecting node embeddings and structural representations. A few recent works have classified node embedding and graph representation methods arguing them to be fundamentally different (e.g., [69]–[72]). Rather, our work shows that these are actually equivalent for downstream classification tasks, with the difference being that one is a Monte Carlo method (embedding) and the other one is deterministic (representation).

Inductive vs Transductive Approaches: Another common misconception our work uncovers, is that of qualifying node embedding methods as transductive learning and graph representation ones as inductive (e.g., [7], [32]). In their original definitions, transductive learning [73], [74], [75] and inductive learning [76], [77] are only to be distinguished on the basis of generalizability of the learned model to unobserved instances. However, this has commonly been misinterpreted as node embeddings methods being transductive and structural representations being inductive. Models which depend solely only on the input feature vectors and the immediate neighborhood structure have been classified as inductive, whereas methods which rely on positional node embeddings to classify relationships in a graph have been incorrectly qualified as transductive.

The confusion seems to be rooted in researchers trying to use a single sample of a node embedding method and failing to generalize. Corollary 4 resolves this confusion by showing that transductive and inductive learning are fundamentally unrelated to positional node embeddings and graph representations. Both node embeddings and structural representations can be inductive if they can detect interesting conceptual patterns or reveal structure in the data. The theory provided by our work strongly adheres to this definition. Our work additionally provides the theoretical foundation behind the performance gains seen by [78] and [79], which employ an ensemble of node embeddings for node classification tasks.

2.5 Conclusions

This chapter provided an invaluable unifying theoretical framework for node embeddings and structural graph representations, bridging methods like SVD and graph neural networks. Using invariant theory, it was shown (both theoretically and empirically) that relationship between structural representations and node embeddings is analogous to that of a distribution and its samples. Moreover, I proved that all tasks that can be performed by node embeddings can also be performed by structural representations and vice-versa. Our empirical results show that node embeddings can be successfully used as inductive learning methods using our framework, and that non-GNN node embedding methods can be significantly more accurate in most tasks than simple GNNs methods. Finally, this chapter introduced new practical guidelines to the use of node embeddings, which we expect will replace today's naïve direct use of node embeddings in graph tasks.

3. LEARNING EXPRESSIVE STRUCTURAL REPRESENTATIONS FOR NOVEL TASKS ON HYPERGRAPHS

The last chapter discussed the connections between graph representation learning strategies. However, graphs can only capture interactions involving pairs of entities (which limits the power of node subset structural representations) whereas in many of the aforementioned domains any number of entities can participate in a single interaction. For example, more than two substances can interact at a specific instance to form a new compound, study groups can contain more that two students, recipes contain multiple ingredients, shoppers purchase multiple items together, etc. Node subset structural representations therefore find their natural habitat in Hypergraphs [80] (see Figure 3.1(a) for example) — which serve as the natural extension of dyadic graphs.

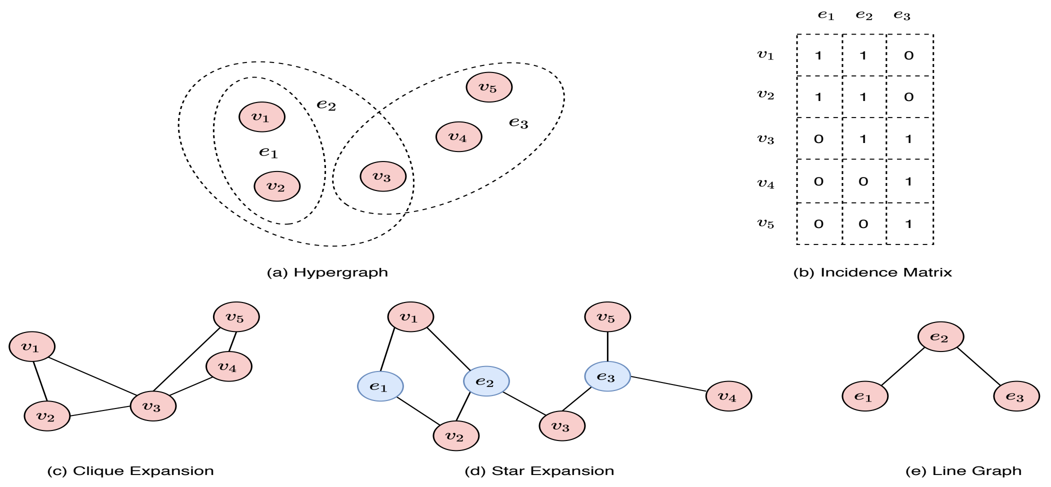


Figure 3.1. A Hypergraph(a) with 5 nodes $v_1, v_2, \ldots v_5$ and 3 hyperedges $e_1 = \{v_1, v_2\}, e_2 = \{v_1, v_2, v_3\}, e_3 = \{v_3, v_4, v_5\}$, its incidence matrix(b), its clique expansion (c), its star expansion (d) and its line graph(e)

Due to the ubiquitous nature of hypergraphs, learning on hypergraphs has been studied extensively for more than a decade [81], [82]. Early works on learning on hypergraphs employed random walk procedures [83]–[85] and the vast majority of them were limited to hypergraphs whose hyperedges have the same cardinality (k-uniform hypergraphs). More recently, with the growing popularity and success of message passing graph neural networks

[2], [7], message passing hypergraph neural networks learning frameworks have been proposed [86]–[90]. These works rely on constructing the clique expansion (Figure 3.1(c)), star expansions (Figure 3.1(d)), or other expansions of the hypergraph that preserve partial information. Subsequently, structural node representations are learned using GNN's on the graph constructed as a proxy of the hypergraph. These strategies are insufficient as either (1) there does not exist a bijective transformation between a hypergraph and the constructed clique expansion (loss of information); (2) they do not accurately model the underlying dependency between a hyperedge and its constituent vertices (for example, a hyperedge may cease to exist if one of the nodes were deleted); (3) they do not directly model the interactions between different hyperedges. The primary goal of this work is to address these issues and to build models which better represent hypergraphs.

Corresponding to the adjacency matrix representation of the edge set of a graph, a hypergraph is commonly represented as an incidence matrix (Figure 3.1(b)), in which a row is a vertex, a column is a hyperedge and an entry in the matrix is 1 if the vertex belongs to the hyperedge. In this work, I directly seek to exploit the incidence structure of the hypergraph to learn representations of nodes and hyperedges. Specifically, for a given hypergraph, I synchronously learn vertex and hyperedge representations that simultaneously take into consideration both the line graph (Figure 3.1(e)) and the set of hyperedges that a vertex belongs to in order to learn provably expressive representations. The jointly learned vertex and hyperedge representations are then used to tackle higher-order tasks such as expansion of partially observed hyperedges and classification of unobserved hyperedges.

While the task of hyperedge (node subset) classification has been studied before, set expansion for relational data has largely been unexplored. For example, given a partial set of substances which are constituents of a single drug, hyperedge expansion entails completing the set of all constituents of the drug while having access to composition to multiple other drugs. A more detailed example for each of these tasks is presented in the Appendix - Section B.1. For the hyperedge expansion task, I propose a GAN framework [91] to learn a probability distribution over the vertex power set (conditioned on a partially observed hyperedge), which maximizes the point-wise mutual information between a partially observed hyperedge and other disjoint vertex subsets in the vertex power set.

The contributions in this chapter can be summarized as: (1) Propose a hypergraph neural network which exploits the incidence structure and hence works on real world sparse hypergraphs which have hyperedges of different cardinalities. (2) Provide provably expressive representations of vertices and hyperedges, as well as that of the complete hypergraph which preserves properties of hypergraph isomorphism. (3) Introduce a new task on hypergraphs – namely the variable sized hyperedge expansion and also perform variable sized hyperedge classification. Furthermore, I demonstrate improved performance over existing baselines on majority of the hypergraph datasets using my proposed model.

3.1 Preliminaries

In my notation for this chapter, I shall use capital case characters (e.g., A) to denote a set or a hypergraph, bold capital case characters (e.g., $\boldsymbol{A}$) to denote a matrix, and capital characters with a right arrow over it (e.g., $\overrightarrow{A}$) to denote a sequence with a predefined ordering of its elements. I shall use lower characters (e.g., a) to denote the element of a set and bold lower case characters (e.g., $\boldsymbol{a}$) to denote vectors. Moreover, I shall denote the i-th row of a matrix $\boldsymbol{A}$ with $\boldsymbol{A}_{\mathrm{i}\cdot}$, the j-th column of the matrix with $\boldsymbol{A}_{\cdot\mathrm{j}}$, and use A_m to denote a subset of the set A of size m i.e., $A_m \subseteq A$; $|A_m| = m$.

Definition 3.1.1 (Hypergraph)**.** *Let $H = (V, E, \boldsymbol{X}, \boldsymbol{E})$ denote a hypergraph H with a finite vertex set $V = \{v_1, \ldots, v_n\}$, corresponding vertex features $\boldsymbol{X} \in \mathbb{R}^{n \times d}$; $d > 0$, a finite hyperedge set $E = \{\mathrm{e}_1, \ldots, \mathrm{e}_m\}$, where $E \subseteq P^*(V) \setminus \{\emptyset\}$ and $\bigcup_{\mathrm{i}=1}^{m} \mathrm{e}_\mathrm{i} = V$, where $P^*(V)$ denotes the power set on the vertices, the corresponding hyperedge features $\boldsymbol{E} \in \mathbb{R}^{m \times d}$; $d > 0$. I use $E(v)$ (termed star of a vertex) to denote the hyperedges incident on a vertex v and use S_H, a set of tuples, to denote the family of stars where $S_H = \{(v, E(v)) : \forall v \in V\}$ called the family of stars of H. When explicit vertex and hyperedge features and weights are unavailable, I will consider $\boldsymbol{X} = \mathbf{1_n}\mathbf{1_n}^T$, $\boldsymbol{E} = \mathbf{1_m}\mathbf{1_m}^T$, where $\mathbf{1}$ represents a $n \times 1$ or $m \times 1$ vector of ones respectively. The vertex and edge set V, E of a hypergraph can equivalently be represented with an incidence matrix $\boldsymbol{I} \in \{0, 1\}^{|V| \times |E|}$, where $\boldsymbol{I}_{\mathrm{ij}} = 1$ if $v_\mathrm{i} \in \mathrm{e}_\mathrm{j}$ and $\boldsymbol{I}_{\mathrm{ij}} = 0$ otherwise. Isomorphic hypergraphs either have the same incidence matrix or a row/column/row and column permutation of the incidence matrix i.e., the matrix $\boldsymbol{I}$ is separately exchangeable. I* *use L_H to denote the line graph (Figure 3.1(e)) of the hypergraph, use $H^\star$ to denote the dual of a hypergraph. Additionally, I define a function LG_H , a multi-valued function termed line hypergraph of a hypergraph - which generalizes the concepts line graph and the dual of a hypergraph and defines the spectrum of values which lies between them. For the scope of this work, I limit myself for LG_H to be a dual valued function - using only the two extremes, such that $LG_H(0) = L_H$ and $LG_H(1) = H^\star$. Also, I use $\Sigma_{n,m}$ to denote the set of all possible attributed hypergraphs H with n nodes and m hyperedges. More formally, $\Sigma_{n,m}$ is the set of all tuples $(V, E, \boldsymbol{X}, \boldsymbol{E})$ — for vertex node set size n and hyperedge set size m.*

Next, I reintroduce 1-WL test for graphs (not hypergraphs) as well as the general form of message passing graph neural networks.

Definition 3.1.2 (1-Weisfeiler-Leman(1-WL) Algorithm)**.** *Let $G = (V, E)$ be a graph, with a finite vertex set V and let $s : V \to \Delta$ be a node coloring function with an arbitrary co-domain Δ and $s(v), v \in V$ denote the color of a node in the graph. Correspondingly, I say a labeled graph (G, s) is a graph G with a complete node coloring $s : V \to \Delta$. The 1-WL Algorithm [92] can then be described as follows: let (G, s) be a labeled graph and in each iteration, $t \geq 0$, the 1-WL computes a node coloring $c_s^{(t)} : V \to \Delta$, which depends on the coloring from the previous iteration. The coloring of a node in iteration $t > 0$ is then computed as $c_s^{(t)}(v) = HASH\left(\left(c_s^{(t-1)}(v), \left\{c_s^{(t-1)}(u) \mid u \in N(v)\right\}\right)\right)$ where HASH is bijective map between the vertex set and Δ, and $N(v)$ denotes the 1-hop neighborhood of node v in the graph. The 1-WL algorithm terminates if the number of colors across two iterations does not change, i.e., the cardinalities of the images of $c_s^{(t-1)}$ and $c_s^{(t)}$ are equal. The 1-WL isomorphism test, is an isomorphism test, where the 1-WL algorithm is run in parallel on two graphs G_1, G_2 and the two graphs are deemed non-isomorphic if a different number of nodes are colored as κ in Δ.*

Definition 3.1.3 (Graph Neural Networks (GNNs))**.** *For a graph $G = (V, E, \boldsymbol{X})$, modern GNNs use the edge connectivity and node features $\boldsymbol{X}$ to learn a representation vector of a node, h_v, or the entire graph, h_G. They employ a neighborhood aggregation strategy, where the representation of a node is iteratively updated by an aggregation of the representations* *of its neighbors. Multiple layers are employed to capture the k-hop neighborhood of a node. The update equation of a GNN layer can be written as*

$$h_v^k = \mathit{COMBINE}(h_v^{k-1}, \mathit{AGGREGATE}^k(\{h_u^{k-1} : u \in N(v)\}))$$

where h_v^k is the representation of node v after k layers and $N(v)$ is the 1-hop neighborhood of v in the graph. [9], [10] *showed that message passing GNNs are no more powerful than the 1-WL algorithm.*

Next, I introduce some group theoretic preliminaries and notions of permutation actions, orbits, equivalence classes and invariant functions specifically in the context of hypergraphs.

Definition 3.1.4 (Finite Symmetric Group $\mathbb{S}_n$)**.** *A finite symmetric group $\mathbb{S}_m$ is a discrete group $\mathcal{G}$ defined over a finite set of size m symbols (w.l.o.g. I shall consider the set $\{1, 2, \ldots, m\}$) and consists of all the permutation operations that can be performed on the m symbols. Since the total number of such permutation operations is $m!$ the order of $\mathbb{S}_m$ is $m!$.*

Definition 3.1.5 (Group Action (left action))**.** *If $\mathcal{A}$ is a set and $\mathcal{G}$ is a group, then $\mathcal{A}$ is a $\mathcal{G}$-set if there is a function $\phi : \mathcal{G} \times \mathcal{A} \to \mathcal{A}$, denoted by $\phi(g, a) \mapsto ga$, such that:*

(i) $1a = a$ for all $a \in \mathcal{A}$, where 1 is the identity element of the group $\mathcal{G}$

(ii) $g(ha) = (gh)(a)$ for all $g, h \in \mathcal{G}$ and $a \in \mathcal{A}$

Definition 3.1.6 (Orbit)**.** *Given a group $\mathcal{G}$ acting on a set $\mathcal{A}$, the orbit of an element $a \in \mathcal{A}$ is the set of elements in $\mathcal{A}$ to which a can be moved by the elements of $\mathcal{G}$. The orbit of a is denoted by $\mathcal{O}(a) = \{g \cdot a | g \in \mathcal{G}\} \subset \mathcal{A}$.*

Definition 3.1.7 (Vertex Permutation action π_V)**.** *A vertex permutation action $\pi \in \mathbb{S}_k$ is the application of a left action $\phi : \mathbb{S}_k \times V_k \to V_k$ with the element π on a sorted sequence of k vertices represented as $\overrightarrow{V_k} = (v_1, \ldots, v_k)$ of a hypergraph to output a corresponding permuted sequence of vertices i.e., $\phi(\pi, \overrightarrow{V_k}) = \overrightarrow{V_{k_\pi}} = (v_{\pi(1)}, \ldots, v_{\pi(k)})$. A permutation action $\pi \in \mathbb{S}_n$ can also act on any vector, matrix, or tensor defined over the nodes V, e.g., $(\boldsymbol{X}_{\mathrm{i}\cdot})_{\mathrm{i} \in V}$, and output an equivalent vector, matrix, or tensor with the order of the nodes permuted e.g., $(\boldsymbol{X}_{\pi(\mathrm{i})\cdot})_{\pi(\mathrm{i}) \in V}$.*

Definition 3.1.8 (Hyperedge Permutation Action π_E)**.** *A hyperedge permutation action $\pi \in \mathbb{S}_k$ is the application of a left action $\psi : \mathbb{S}_k \times E_k \to E_k$ with the element π on a sorted sequence of m hyperedges represented as $\overrightarrow{E_k} = (\mathrm{e}_1, \ldots, \mathrm{e}_k)$ of a hypergraph to output a corresponding permuted sequence of hyperedges i.e., $\psi(\pi, \overrightarrow{E_k}) = \overrightarrow{E_{k_\pi}} = (\mathrm{e}_{\pi(1)}, \ldots, \mathrm{e}_{\pi(k)})$. A permutation action $\pi \in \mathbb{S}_m$ can also act on any vector, matrix, or tensor defined over the hyperedges E, e.g., $(\boldsymbol{E}_{\mathrm{i}\cdot})_{\mathrm{i}\in E}$, and output an equivalent vector, matrix, or tensor with the order of the hyperedges permuted e.g., $(\boldsymbol{E}_{\pi(\mathrm{i})\cdot})_{\pi(\mathrm{i})\in E}$.*

It is crucial to note that a vertex permutation action can be simultaneously performed along with the hyperedge permutation. I represent a joint permutation on the entire edge set E as $\pi_2(\pi_1(E))$, and for a hyperedge $\mathrm{e} \in E$ as $\pi_2(\pi_1(\mathrm{e}))$ where $\pi_\mathrm{i} \in \mathbb{S}_n, \pi_2 \in \mathbb{S}_m$

Definition 3.1.9 (Node Equivalence Class/ Node Isomorphism)**.** *The equivalence classes of vertices $v \in V$ of a hypergraph H under the action of automorphisms between the vertices are called vertex equivalence classes or vertex orbits. If two vertices $v_1, v_2 \in V$ are in the same vertex orbit, I say they are node isomorphic and are denoted by $v_1 \cong v_2$.*

Definition 3.1.10 (Hyperedge Orbit/ Isomorphism)**.** *The equivalence classes of non empty subsets of vertices $\mathrm{e} \in \mathcal{P}^\star(V)\setminus\emptyset$; $\mathrm{e} \in E$ of a hypergraph G under the action of automorphisms between the subsets are called hyperedge orbits. If two proper subsets $\mathrm{e}_1, \mathrm{e}_2 \in P^*(V)\setminus\emptyset$ are in the same hyperedge orbit, we say they are hyperedge isomorphic and are denoted by $\mathrm{e}_1 \cong \mathrm{e}_2$.*

Definition 3.1.11 (Hypergraph Orbit and Isomorphism)**.** *The hypergraph orbit of a hypergraph H, given by the application of the elements π of the finite symmetry group $\mathbb{S}_n$ on the vertex set V/ $\mathbb{S}_m$ on the edge set E/ or any combination of the two and appropriately modifying the associated matrices $\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{I}$ of the hypergraph. Two hypergraphs H_1 and H_2 are said to be isomorphic (equivalent) denoted by $H_1 \cong H_2$ iff there exists either a vertex permutation action or hyperedge permutation action or both such that $H_1 = \pi \cdot H_2$. The hypergraph orbits are then the equivalence classes under this relation; two hypergraphs H_1 and H_2 are equivalent iff their hypergraph orbits are the same.*

Definition 3.1.12 ($\mathcal{G}$-invariant functions)**.** *A function ϕ acting on a hypergraph H given by $\phi : \Sigma_{n,m} \to \mathbb{R}^\circ$ is $\mathcal{G}$-invariant whenever it is invariant to any vertex permutation/*

edge permutation action $\boldsymbol{\pi} \in \mathbb{S}.$ in the $\Sigma_{n,m}$ symmetric space i.e., $\phi(\boldsymbol{\pi} \cdot H) = \phi(H)$ and all isomorphic hypergraphs obtain the same representation. Similarly, a function $\rho : P^*(V) \backslash \emptyset \to R^{\cdot}$ acting on a hyperedge for a given hypergraph H, is said to be $\mathcal{G}$-invariant iff all isomorphic hyperedges obtain the same representation.

Task Descriptions:

I now provide, the descriptions of the tasks which I introduce on hypergraphs which primarily rely on structural representations of node subsets.

Partially observed hyperedge expansion: Consider a hypergraph $H = (V, E', \boldsymbol{X}, \boldsymbol{E})$ where a small fraction of hyperedges in the hyperedge set are partially observed and let E be the completely observed hyperedge set. A partially observed hyperedge implies $\exists \mathrm{e}'_{\mathrm{i}} \in E'$, $\exists v_{\mathrm{j}} \in V, v_{\mathrm{j}} \notin \mathrm{e}'_{\mathrm{i}}$ but $v_{\mathrm{j}} \in \mathrm{e}_{\mathrm{i}}$, $\mathrm{e}_{\mathrm{i}} \in E$, where e_{i} is the corresponding completely observed hyperedge of e'_{i} The task here is, given a partial hyperedge $\mathrm{e}' \in E', \mathrm{e}' \notin E$ but $\mathrm{e}' \subset \mathrm{e}, \mathrm{e} \in E$, to complete e' with vertices from V so that after hyperedge expansion $\mathrm{e}' = \mathrm{e}$.

Unobserved hyperedge classification: Consider a hypergraph $H = (V, E', \boldsymbol{X}, \boldsymbol{E}')$ with an incompletely observed hyperedge set E' and let E be the corresponding completely observed hyperedge set with $E' \subset E$. An incomplete hyperedge set implies $\exists \mathrm{e} \in E$; $\mathrm{e} \notin E'$ where $|E'| < |E| = m$. It is important to note that in this case, if a certain hyperedge is present in E', then the hyperedge is not missing any vertices in the observed hyperedges. The task here is, for a given hypergraph H, to predict whether a new hyperedge e was present but unobserved in the noisy hyperedge set i.e., $\mathrm{e} \notin E'$ but $\mathrm{e} \in E$.

3.2 Theory

Previous works on hypergraph neural networks [86]–[88], employ a proxy graph to learn vertex representations for every vertex $v \in V$, by aggregating information over its neighborhood. Hyperedge representations (or alternatively, hypergraph) are then obtained, when necessary, by using a pooling operation (e.g. sum, max, mean, etc) over the vertices in the hyperedge (vertex set of the hypergraph). However, such a strategy, fails to (1) preserve properties of equivalence classes of hyperedges/hypergraphs and (2) capture the implicit higher order interactions between the nodes/ hyperedges, and fails on higher order tasks as shown by [11].

To alleviate these issues, in this work, I use a message passing framework on the incidence graph representation of the observed hypergraph, which synchronously updates the node and observed hyperedge embeddings as follows:

$$\begin{split}\boldsymbol{h}_{\mathrm{e}}^{k} = \sigma(\boldsymbol{W}_{E}^{k} \cdot (\boldsymbol{h}_{\mathrm{e}}^{(k-1)} \otimes f^{k}(\{(\boldsymbol{h}_{v}^{(k-1)} \otimes p^{k}(\{\boldsymbol{h}_{\mathrm{e}'}^{(k-1)} \\ : \mathrm{e}' \ni v\})) : \forall v \in \mathrm{e} \,\mathrm{where}\, v \in V, \mathrm{e}, \mathrm{e}' \in E\}))) \end{split} \tag{3.1}$$

$$\begin{split}\boldsymbol{h}_{v}^{k} = \sigma(\boldsymbol{W}_{V}^{k} \cdot (\boldsymbol{h}_{v}^{(k-1)} \otimes g^{k}(\{\boldsymbol{h}_{\mathrm{e}}^{(k-1)} \otimes q^{k}(\{\boldsymbol{h}_{v'}^{(k-1)} \\ : v' \in \mathrm{e}\})) : \forall \mathrm{e} \ni v \;\; \mathrm{where}\, v, v' \in V, \mathrm{e} \in E\}))) \end{split} \tag{3.2}$$

where, $\otimes$ denotes vector concatenation, f^k, g^k, p^k, q^k are injective set, multiset functions (constructed via [27], [28], [93]) in the k^{th} layer, $\boldsymbol{h}_{\mathrm{e}}^{k}, \boldsymbol{h}_{v}^{k}$ are the vector representations of the hyperedge and vertices after k layers, $\boldsymbol{W}_{V}^{k}, \boldsymbol{W}_{E}^{k}$ are learnable weight matrices and σ is an element-wise activation function. I use K (in practice, K=2) to denote the total number of convolutional layers used. From Equation (3.2) it is clear to see that a vertex not only receives messages from the hyperedges it belongs to, but also from neighboring vertices in the clique expansion. Similarly, from Equation (3.1), a hyperedge receives messages from its constituent vertices as well as neighboring hyperedges in the line graph of the hypergraph.

However, the above equations, standalone do not present a framework to learn representations of unobserved hyperedges for downstream tasks. In order to do this, post the convolutional layers, the representation of any hyperedge (observed or unobserved) are obtained using a function $\Gamma : P^{\star}(V) \times \Sigma_{n,m} \to \mathbb{R}^d$ as:

$$\begin{split}\Gamma(\mathrm{e}', H; \Theta) = \phi(\{h_{v_{\mathrm{i}}}^{K} : v_{\mathrm{i}} \in \mathrm{e}'\}) \otimes \rho(\{\boldsymbol{h}_{\mathrm{e}}^{K} : \mathrm{e} \ni v_{\mathrm{i}} \, \forall v_{\mathrm{i}} \in \mathrm{e}'\}) \\ \mathrm{where}\, v_{\mathrm{i}} \in V, \mathrm{e} \in E' \end{split} \tag{3.3}$$

where ϕ, ρ are injective set, multiset functions respectively, and Θ denotes the model parameters of the entire hypergraph neural network (convolutional layers, set functions). Correspondingly, the representation of the complete hypergraph is obtained using a function $\Gamma : \Sigma_{n,m} \to \mathbb{R}^d$ as:

$$\Gamma(H;\Theta) = \phi(\{h_v^K : v \in V\}) \otimes \rho(\{h_{\mathrm{e}}^K : \mathrm{e} \in E\})) \tag{3.4}$$

In what follows, I list the properties of the vertex/ hyperedge representations. All proofs are presented in the Supplementary Material (arXiv version) [94].

Property 3.2.1 (Vertex Representations)**.** *The representation of a vertex $v \in V$ in a hypergraph H learnt using Equation (3.2) is a $\mathcal{G}$-invariant representation $\Phi(v, V, E, \boldsymbol{X}, \boldsymbol{E})$ where $\Phi : V \times \Sigma_{n,m} \to \mathbb{R}^d, d \geq 1$ such that*

$$\Phi(v, V, E, \boldsymbol{X}, \boldsymbol{E}) = \Phi((\pi_1(v), \pi_1(V), \pi_2(\pi_1(E)), \pi_1(\boldsymbol{X}), \pi_2(\pi_1(\boldsymbol{E})))$$

$\forall \pi_1 \forall \pi_2$ where $\pi_1 \in \mathbb{S}_n$ and $\pi_2 \in \mathbb{S}_m$. Moreover, two vertices v_1, v_2 which belong to the same vertex equivalence class i.e. $v_1 \cong v_2$ obtain the same representation.

Property 3.2.2 (Hyperedge Representations)**.** *The representation of a hyperedge $\mathrm{e} \in E$ in a hypergraph H learnt using Equation (3.1) is a $\mathcal{G}$-invariant representation $\Phi(\mathrm{e}, V, E, \boldsymbol{X}, \boldsymbol{E})$ where $\Phi : P^\star(V) \times \Sigma_{n,m} \to \mathbb{R}^d, d \geq 1$ such that*

$$\Phi(\mathrm{e}, V, E, \boldsymbol{X}, \boldsymbol{E}) = \Phi((\pi_2(\pi_1(\mathrm{e})), \pi_1(V), \pi_2(\pi_1(E)), \pi_1(\boldsymbol{X}), \pi_2(\pi_1(\boldsymbol{E})))$$

$\forall \pi_1 \forall \pi_2$ where $\pi_1 \in \mathbb{S}_n$ and $\pi_2 \in \mathbb{S}_m$ Moreover, two hyperedges $\mathrm{e}_1, \mathrm{e}_2$ which belong to the same hyperedge equivalence class i.e. $\mathrm{e}_1 \cong \mathrm{e}_2$ obtain the same representation.

Next, I restate a theorem from [95] which provides a means to deterministically distinguish non isomorphic hypergraphs. Subsequently, I characterize the expressivity of my model to distinguish non-isomorphic hypergraphs.

Theorem 3.2.1 ([95])**.** *Let H_1, H_2 be hypergraphs without isolated vertices whose line hypergraphs LG_{H_1}, LG_{H_2} are isomorphic. Then $H_1 \cong H_2$ if and only if there exists a bijection $\beta : VLG_{H_1} \to VLG_{H_2}$ such that $\beta(S_{H_1}) = S_{H_2}$, where $S_{H_{\mathrm{i}}}$ is the family of stars of the hypergraph H_{i}*

Theorem 3.2.2. *Let H_1, H_2 be two non isomorphic hypergraphs with finite vertex and hyperedge sets and no isolated vertices. If the Weisfeiler-Lehman test of isomorphism decides their line graphs L_{H_1}, L_{H_2} and the star expansions of their duals $H_1^\star, H_2^\star$ to be not isomorphic then there exists a function $\Gamma : \Sigma_{n,m} \rightarrow \mathbb{R}^d$ (via Equation (3.4)) and parameters Θ that maps the hypergraphs H_1, H_2 to different representations.*

I now, extend this to the expressivity of the hyperedge representations and then show that the property of separate exchangeability [96] of the incidence matrix is preserved by the hypergraph representation.

Corollary 6. *There exists a function $\Gamma : P^\star(V) \times \Sigma_{n,m} \rightarrow \mathbb{R}^d$ (via Equation (3.3)) and parameters Θ that maps two non-isomorphic hyperedges $\mathrm{e}_1, \mathrm{e}_2$ to non identical representations.*

Remark 3 (Separate Exchangeability)**.** *The representation of a hypergraph H learnt using the function $\Gamma : \Sigma_{n,m} \rightarrow \mathbb{R}^d$ (via Equation (3.4)) preserves the separate exchangeability of the incidence structure $\boldsymbol{I}$ of the hypergraph.*

3.3 Learning Framework

I now describe the learning procedures for the two tasks, namely variable size hyperedge classification and variable size hyperedge expansion.

3.3.1 Hyperedge Classification

For a hypergraph H, let E' denote the partially observed hyperedge set in my data corresponding to the true hyperedge set E. The goal here is to learn a classifier $r : \mathbb{R}^d \rightarrow \mathbb{R}$ over the representations of hyperedges (obtained using Equation (3.3)) *s.t* $\sigma(r(\Gamma(\{v_\mathrm{i}, v_2, \ldots, v_M\}, H)))$ is used to classify if an unobserved hyperedge $\mathrm{e} = \{v_1, v_2, \ldots, v_M\}$ exists i.e. $\mathrm{e} \notin E'$ but $\mathrm{e} \in E$ where all $v_\mathrm{i} \in V$ for $\mathrm{i} \in \{1, 2, \ldots, M\}$, and σ is the logistic sigmoid.

Now, for the given hypergraph H, let $\boldsymbol{Y}^H \in \{0,1\}^{|P^\star(V) \backslash \emptyset|}$ be the target *random variables* associated with the vertex power set of the graph. Let $\mathcal{Y}^H \in \{0,1\}^{|P^\star(V) \backslash \emptyset|}$ be the corresponding true values attached to the vertex subsets in the power set, such that $\mathcal{Y}^H_\mathrm{e} = 1$ iff

$\mathrm{e} \in E$. Next, I model the joint distribution of the hyperedges in the hypergraphs by making a mean field assumption as:

$$P(H) = \prod_{\mathrm{e} \in P^{\star}(V) \setminus \emptyset} \text{Bernoulli}(\boldsymbol{Y}_{\mathrm{e}}^{G} = \mathcal{Y}_{\mathrm{e}}^{G} | r(\Gamma(\mathrm{e}, H); \Theta)) \tag{3.5}$$

Subsequently, to learn the model parameters Θ - I make a closed world assumption and treat only the observed hyperedges in E' as positive and all other edge as false and seek to maximize the log-likelihood.

$$\begin{aligned} \Theta = \arg\max_{\Theta} \sum_{\mathrm{e} \in E'} \log p(\boldsymbol{Y}_{\mathrm{e}}^{H} = 1 | r(\Gamma(\mathrm{e}, H)); \Theta) + \\ \sum_{\mathrm{e} \in P^{\star}(V) \setminus \{E', \emptyset\}} \log p(\boldsymbol{Y}_{\mathrm{e}}^{H} = 0 | r(\Gamma(\mathrm{e}, H)); \Theta) \end{aligned} \tag{3.6}$$

Since the size of vertex power set ($2^{|V|}$), grows exponentially with the number of vertices, it is computationally intractable to use all negative hyperedges in the training procedure. My training procedure, hence employs a negative sampling procedure (in practice, I use 5 distinct negative samples for every hyperedge in every epoch) combined with a cross entropy loss function, to learn the model parameters via back-propagation. This framework can trivially be extended to perform multi class classification on variable sized hyperedges.

3.3.2 Hyperedge Completion

The set expansion task introduced in [28] makes the infinite de-Finetti assumption i.e. the elements of the set are *i.i.d.* When learning over finite graphs and hypergraphs, this assumption is no longer valid - since the data is relational - i.e. a finite de-Finetti [97] assumption is required. Additionally, the partial exchangeability of the structures (adjacency matrix/ incidence matrix) [96] have to be factored in as well.

This raises multiple concerns: (1) computing mutual information of a partial vertex set with all other disjoint vertex subsets in the power set is computationally intractable; (2) to learn a model in the finite de-Finetti setting, we need to consider all possible permutations for a vertex subset. For example, under the conditions of finite exchangeability, the point-wise mutual information between two random variables X, Y - where both are disjoint elements of the vertex power set (or hyperedges) i.e. $X, Y \in P^{\star}(V) \setminus \emptyset$, $X \cap Y = \emptyset$ is given by:

$$s(X|Y) = \log p(X \cup Y \,|\, \alpha) - \log p(X \,|\, \alpha) p(Y \,|\, \alpha) \tag{3.7}$$

where α is a prior and each of $p(X|\alpha), p(Y|\alpha), P(X \cup Y|\alpha)$ cannot be factorized any further i.e.

$$p(X|\alpha) = \frac{1}{|X|!} \sum_{\pi \in \Pi_X} \log p(v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(|X|)} \,|\, \alpha) \tag{3.8}$$

where $v_{\mathrm{i}} \in X, \mathrm{i} \in \{1, 2, \ldots, |X|\}$ and Π_X denotes the set of all possible permutations of the elements of X. The inability to factorize Equation (3.8) any further, leaves no room for any computational benefits by a strategic addition of vertices - one at a time (i.e. no reuse of computations, whatsoever).

As a solution to this computationally expensive problem, I propose a GAN framework [91] to learn a probability distribution over the vertex power set, conditioned on a partially observed hyperedge, without sacrificing on the underlying objective of maximizing point-wise mutual information between X, Y (Equation (3.7)). I describe the working of the generator and the discriminator of the GAN, with the help of a single example below.

Let $\overline{\mathrm{e}}$ denote a partially observed hyperedge and $\Gamma(\overline{\mathrm{e}}, G)$ denote the representation of the partially observed hyperedge obtained via Equation (3.3). Let $V_K, \overline{V_K}$ denote the true and predicted vertices respectively to complete the partial hyperedge $\overline{\mathrm{e}}$, where $V_K, \overline{V_K} \subseteq V \setminus \{\overline{\mathrm{e}}\}$.

Generator($G^{\star}$)

The goal of the generator is to accurately predict $\overline{V_K}$ as V_K. I solve this using a two-fold strategy - first predict the number of elements K, missing in the partially observed hyperedge $\overline{\mathrm{e}}$ and then jointly select K vertices from $V \setminus \overline{\mathrm{e}}$. Ideally, the selection of the best K vertices should be performed over all vertex subsets of size K (where vertices are sampled from $V \setminus \overline{\mathrm{e}}$ without replacement). However, this is computationally intractable even for small values e.g., $K = 2, 3$ for large graphs with millions of nodes.

I predict the number of elements missing in a hyperedge, K, using a function $a_1 : \mathbb{R}^d \rightarrow \mathbb{N}$ over the representation of the partial hyperedge, $\Gamma(\overline{\mathrm{e}}, G)$. To address the problem of jointly selecting a set of k vertices without sacrificing on computational tractability, I seek to employ a variant of the Top-K problem often used in computing literature.

The standard top-K operation can be adapted to vertices as: given a set of vertices of a graph $\{v_1, v_2, \cdots v_n\} = V \setminus \{\bar{\mathrm{e}}\}$, to return a vector $A = [A_1, \ldots, A_n]^\top$ such that

$$A_{\mathrm{i}} = \begin{cases} 1, & \text{if } v_{\mathrm{i}} \text{ is a top- } K \text{ element in } V \setminus \bar{\mathrm{e}} \\ 0, & \text{otherwise.} \end{cases}$$

However a standard top-K procedure, which operates by sampling vertices (from the vertex set - a categorical distribution) is discrete and hence not differentiable. To alleviate the issue of differentiability, Gumbel softmax [98], [99] could be employed to provide a differentiable approximation to sampling discrete data. However, explicit top-K Gumbel sampling (computing likelihood for all possible sets of size k over the complete domain) is computationally prohibitive and hence finds limited applications in hypergraphs with a large number of nodes and hyperedges.

In this work, I sacrifice on exact differentiability and focus on scalability. I limit the vertex pool (which can complete the hyperedge) to only vertices in the two hop neighborhood (in the clique expansion C_H) of the vertices in the partial hyperedge. For real world datasets, even the reduced vertex pool consists of a considerable number of vertices - and explicitly computing all sets of size K is still prohibitive. In such cases, I sample uniformly at random a large number of distinct vertex subsets of size k from the reduced vertex pool, where k is the size predicted by the generator. In practice, the large number is typically min($\binom{P}{k}$, 100,000), where P is the number of vertices in the reduced vertex pool. Subsequently, I compute the inner product of the representations of these subsets (computed using Equation (3.3)) with the representation of the partially observed hyperedge. I then use a simple Top-1 to select the set of size k which maximizes the inner product.

Discriminator($D^\star$)

The goal of the discriminator is to distinguish the true, but unobserved hyperedges from the others. To do this, I obtain representations of $\Gamma(\bar{\mathrm{e}}, G), \Gamma(V_K, G), \Gamma(\bar{\mathrm{e}} \cup V_K, G)$ (and similarly for the predicted $\overline{V_K}$ using the generator $G^\star$) and employ the discriminator in the same vein as Equation (3.7). As a surrogate for the log-probablities, I learn a function $g : \mathbb{R}^d \rightarrow \mathbb{R}^d$ over the representations of $\Gamma(\bar{\mathrm{e}}, G), \Gamma(V_K, G), \Gamma(\bar{\mathrm{e}} \cup V_K, G)$ (log-probabilities in higher dimensional space). Following this, I apply a function $f : \mathbb{R}^d \rightarrow \mathbb{R}^+ \cup \{0\}$, as a surrogate for the mutual information computation. The equation of discriminator can then be listed as:

$$\begin{aligned} D^\star(V_K|\bar{\mathrm{e}}, H) = \sigma(f(g(\Gamma(\bar{\mathrm{e}} \cup V_K, H)) \\ -g(\Gamma(\bar{\mathrm{e}}, H)) - g(\Gamma(V_K, H)))) \end{aligned} \tag{3.9}$$

and correspondingly for $D^\star(G^\star(\overline{V_K}|\bar{\mathrm{e}}, G))$, where σ is the logistic sigmoid.

My training procedure for the GAN, over the hypergraph H, can then be summarized as follows. Let $V^\dagger$ denote the value function and let E' denote a set of partial hyperedges and E denote the corresponding set with all hyperedges completed. Let $V_{K_{\bar{\mathrm{e}}}}, \overline{V_{K_{\bar{\mathrm{e}}}}}$ denote the corresponding true and predicted vertices to complete the hyperedge. The value function can then be written as:

$$\begin{aligned} \min_{G^\star} \max_{D^\star} V^\dagger(D^\star, G^\star) = \sum_{\mathrm{e}' \in E'} \log D^\star(V_{K_{\mathrm{e}'}}|\mathrm{e}', H) + \\ \log(1 - D^\star(G^\star(\mathrm{e}', H))) \end{aligned} \tag{3.10}$$

In practice, the model parameters of the GAN are learnt using a cross entropy loss and back-propagation. An MSE loss is employed to train the function a_1, the function that predicts the number of missing vertices in a hyperedge, using ground truth information about the number of missing vertices in the partial hyperedge.

3.4 Results

I first briefly describe the datasets and then present our experimental results on the two hypergraph tasks.

Datasets

I use the publicly available hypergraph datasets from [100] to evaluate the proposed models against multiple baselines (described below). I ignore the timestamps in the datasets and only use unique hyperedges which contain greater than 1 vertex. Moreover, none of the datasets have node or hyperedge features. I summarize the dataset statistics in the Supplementary material. I briefly describe the hypergraphs and the hyperedges in the different datasets below.

- Online tagging data (tags-math-sx; tagsask-ubuntu). In this dataset, nodes are tags (annotations) and a hyperedge is a set of tags for a question on online Stack Exchange forums.
- Online thread participation data (threads-math-sx; threads-ask-ubuntu): Nodes are users and a hyperedge is a set of users answering a question on a forum.
- Two drug networks from the National Drug Code Directory, namely (1) NDC-classes: Nodes are class labels and a hyperedge is the set of class labels applied to a drug (all applied at one time) and (2) NDC-substances: Nodes are substances and a hyperedge is the set of substances in a drug.
- US. Congress data (congress-bills): Nodes are members of Congress and a hyperedge is the set of members in a committee or cosponsoring a bill.
- Email networks (email-Enron; email-Eu): Nodes are email addresses and a hyperedge is a set consisting of all recipient addresses on an email along with the sender's address.
- Contact networks (contact-high-school; contact-primary-school): Nodes are people and a hyperedge is a set of people in close proximity to each other.
- Drug use in the Drug Abuse Warning Network (DAWN): Nodes are drugs and a hyperedge is the set of drugs reportedly used by a patient before an emergency department visit.

Experimental Results

Hyperedge Classification

In this task, I compare my model against five baselines. The first is a trivial predictor, which always predicts 1 for any hyperedge (in practice, I use 5 negative samples for every real hyperedge). The second two baselines utilize a GCN [2] or GraphSAGE [7] on the clique expansion of the hypergraph. GCN on the clique expansion on the hypergraph is the model proposed by [86] as HGNN. For the fourth baseline, I utilize the star expansion of the hypergraph - and employ a heterogeneous RGCN to learn the vertex, hyperedge embeddings.

Table 3.1. F1 scores for the hyperedge classification task (Higher is better).

	Trivial	Clique Expansion-GCN (HGNN)	Clique Expansion-SAGE	Star Expansion - Heterogenous - RGCN	Ours
NDC-classes	0.286	0.614(0.005)	0.657(0.020)	0.676(0.049)	**0.768(0.004)**
NDC-substances	0.286	0.421(0.014)	0.479(0.007)	**0.525(0.006)**	**0.512(0.032)**
DAWN	0.286	0.624(0.010)	0.664(0.006)	0.634(0.003)	**0.677(0.004)**
contact-primary-school	0.286	0.645(0.031)	0.681(0.014)	0.669(0.012)	**0.716(0.034)**
contact-high-school	0.286	**0.759(0.030)**	0.724(0.009)	0.739(0.012)	**0.786(0.033)**
tags-math-sx	0.286	0.599(0.009)	**0.635(0.003)**	0.572(0.003)	**0.642(0.006)**
tags-ask-ubuntu	0.286	0.545(0.005)	0.597(0.007)	0.545(0.006)	**0.605(0.002)**
threads-math-sx	0.286	0.453(0.017)	0.553(0.012)	0.487(0.006)	**0.586(0.002)**
threads-ask-ubuntu	0.286	0.425(0.007)	**0.512(0.007)**	0.464(0.010)	0.488(0.012)
email-Enron	0.286	0.618(0.032)	0.594(0.046)	0.599(0.040)	**0.685(0.016)**
email-EU	0.286	0.664(0.003)	0.651(0.019)	0.661(0.006)	**0.687(0.002)**
congress-bills	0.286	0.412(0.003)	0.530(0.055)	0.544(0.004)	**0.566(0.011)**

[1] A 5-fold cross validation procedure is used - numbers outside the parenthesis are the mean values and the standard deviation is specified within the parenthesis
[2] Bold values show maximum empirical average, and multiple bolds happen when its standard deviation overlaps with another average.

Table 3.2. Normalized Set Difference scores for the hyperedge expansion task (lower is better)

	Simple	Recursive	Ours
NDC-classes	1.207(0.073)	1.163(0.015)	**1.107(0.007)**
NDC-substances	1.167(0.000)	**1.161(0.009)**	**1.153(0.004)**
DAWN	1.213(0.006)	1.197(0.022)	**1.088(0.018)**
contact-primary-school	0.983(0.006)	0.986(0.001)	**0.970(0.005)**
contact-high-school	**0.990(0.014)**	1.000(0.000)	**0.989(0.001)**
tags-math-sx	1.012(0.025)	1.003(0.014)	**0.982(0.011)**
tags-ask-ubuntu	1.008(0.003)	1.005(0.003)	**0.972(0.001)**
threads-ask-ubuntu	0.999(0.000)	0.999(0.000)	**0.981(0.003)**
email-Enron	1.152(0.045)	1.182(0.015)	**1.117(0.049)**
email-EU	1.199(0.002)	1.224(0.010)	**1.116(0.013)**
congress-bills	1.186(0.004)	1.189(0.001)	**1.107(0.004)**

[1] A 5-fold cross validation procedure is used - numbers outside the parenthesis are the mean values and the standard deviation is specified within the parenthesis
[2] Bold values show minimum empirical average, and multiple bolds happen when its standard deviation overlaps with another average.

In each of the baselines, unobserved hyperedge embeddings are obtained by aggregating the representations of the vertices it contains, using a learnable set function [27], [28]. I report F1 scores on the eight datasets in Table 3.1. More details about the experimental setup is presented in the Supplementary material (arXiv version) [94].

Hyperedge Expansion

Due to lack of prior work in hyperedge expansion, here I compare my strategy against two other baselines for hyperedge expansion (with the an identical GAN framework and setup to predict the number of missing vertices, albeit without computing joint representations of predicted vertices) : (1) Addition of Top-K vertices, considered independently of each other (2) Recursive addition of Top-1 vertex. Since all the three models are able to accurately (close to 100% accuracy) predict the number of missing elements, I introduce *normalized set difference*, as a statistic to compare the models. Normalized Set difference (permutation invariant) is given by the number of insertion/ deletions/ modifications required to go from the predicted completion to the target completion divided by the number of missing elements in the target completion. For example, let {7,8,9} be a set which we wish to expand. Then the normalized set difference between a predicted completion {3,5,1,4} and target completion {1,2} is computed as by (1+2)/2 = 1.5 (where there is 1 modification and 2 deletions). It is clear to see that, a lower normalized set difference score is better and a score of 0 indicates a perfect set prediction. Results are presented in Table 3.2.

In the hyperedge classification task, from Table 3.1 it is clear to see that the proposed model model which with provable expressive properties performs better than the baselines, on most datasets. All three non-trivial baselines appear to suffer from their inability to capture higher order interactions between the vertices in a hyperedge. Moreover, the loss in information by using a proxy graph - in the form of the clique expansion - also affects the performance of the SAGE and GCN baselines. The SAGE baseline obtaining better F1 scores over GCN suggests that the self loop introduced between vertices in the clique expansion appears to hurt performance. The lower scores of the star expansion models can be attributed to its inability in capturing vertex-vertex and hyperedge-hyperedge interactions.

For the hyperedge expansion task, from Table 3.2 it is clear to see that adding vertices in a way which captures interactions amongst them performs better than adding vertices independently of each other or in a recursive manner. The relatively weaker performance of adding vertices recursively, one at a time can be attributed to a poor choice of selection of the first vertex to be added (once an unlikely vertex is added, the sequence cannot be corrected).

3.5 Conclusions

In this chapter, I developed a hypergraph neural network to learn provably expressive representations of vertices, hyperedges and the complete hypergraph. I proposed frameworks for hyperedge classification and a novel hyperedge expansion task, evaluated performance on multiple real-world hypergraph datasets, and demonstrated consistent, significant improvement in accuracy, over state-of-the-art models.

4. CONDITIONAL INVARIANCES FOR CONFORMER INVARIANT PROTEIN REPRESENTATION LEARNING

In this chapter, I look at data, which are represented as graphs for the sake of convenience - but exhibit symmetries more than that of permutation equivariance (invariance). Specifically, I look at the case of protein molecules.

Learning on proteins (and 3D macromolecular structures in general) is a rapidly growing application area in geometric deep learning, [101], [102] which presents itself with challenging domain specific information but a seemingly boundless number of real world applications [103]. Traditionally, proteins have been modeled as either 3D images or graphs and subsequently standard 3D CNNs [104], [105], graph neural networks (GNNs) [2], [7], transformers [106] have been employed to learn low dimension embeddings of the protein which are then used for downstream tasks. More recently, several works [107]–[110] have enriched the aforementioned standard neural network models, with inductive biases such as being equivariant (invariant) to transformations from the Euclidean and rotation groups. While equivariance (invariance) to the transformations in these groups are necessary properties for the model, unfortunately, they are limited to only capture rigid transformations of the input object.

Proteins, however, are not rigid structures and different proteins have multiple different conformations [13], [14]. However, in nearly all publicly available datasets, only one conformer per protein is observed in the dataset. Current models therefore could lead to the different conformers of the same protein obtaining different representations, where they should get the same representation. In light of this limitation, a question naturally arises: *Is it possible to learn conformer-invariant protein representations?*

The literature on geometric deep learning has achieved much success with neural network that explicitly model equivariances (invariances) to group transformations [2], [57], [111]–[113]. Among applications to physical sciences, group equivariant graph neural networks and transformers have specifically found applications to small molecules, as well as larger molecules such as proteins with tremendous success [107]–[110], [114]–[119]. However, these models do not yet account for all invariances. The transformations they are invariant to do not depend on the input. For instance, invariance to the Euclidean group restricts the

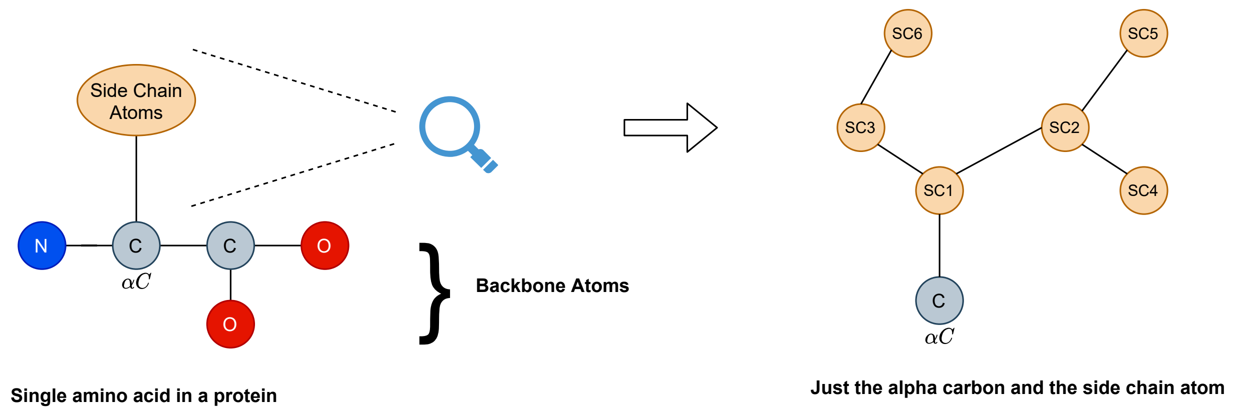

Side Chain Atoms
N
C
αC
C
O
O
Backbone Atoms
Single amino acid in a protein
SC6
SC5
SC3
SC2
SC1
SC4
C
αC
Just the alpha carbon and the side chain atom

Figure 4.1. Magnified image of side chain of a single generic amino acid (with 6 atoms in the side chain with a specified connectivity) in a protein molecule. A protein molecule typically contains tens to hundreds of amino acids

protein representation to act as if the protein were a rigid structure, regardless of the protein under consideration.

Our Approach: I propose a representation method where the set of symmetries in the task are input-dependent, which I denote *conditional symmetries*. For my specific application, I model every protein as a directed forest, where every amino acid forms a directed tree. I leverage the constructed directed forest of the protein to sample viable protein conformations (where viability is checked with Protein structure validation tools such as Molprobity [120]–[122]), which is additionally coupled with an MCMC framework used to create data augmentations to train the neural network to learn conformer invariant representations. The directed tree associated with an amino acid allows us to transform the positional coordinates associated with atoms in the side chain (Figure 4.1) - where a node can be transformed about its immediate parents or any of its ancestors.

Our **contributions** can be summarized as follows:

- I provide a principled strategy to sample conformations for any given protein from the support of its protein conformer distribution (which consists of all its viable protein conformations) which captures their true flexibility [13], [14]. Viable conformations respect domain specific constraints such as those on dihedral angles, steric repulsions, among others. This is particularly useful in the scenario when the set of viable transformations is different across different proteins.

- Further, I develop a Markov chain Monte Carlo learning framework which guides the sampling of protein conformations in such a way – so as to provide theoretical guarantees that the representations obtained by all conformations of a protein are identical.
- I then provide a few guiding principles of conditional invariant representations as inductive biases, which serves as a generalization of group invariant neural networks.
- Finally, I perform experimental evaluation of the proposed approach, where endowing baseline models with our proposed strategy shows noticeable improvements on four different tasks on proteins.

4.1 Conditional Invariances for Proteins

The symmetry of an object is the set of transformations that leaves the object invariant. The notion of symmetries, expressed through the action of a group on functions defined on some domain, has served as a fundamental concept of geometric deep learning. In this section, I start by defining the concept of input-dependent *conditional symmetries* and then then proceed to specifically tailor these conditional symmetries that are both computationally efficient and useful for representing protein conformations. Some group theoretic preliminaries are presented in Section C.1.

Definition 4.1.1 (Conditionally symmetric-invariant functions)**.** *A function $f : \Omega \to \mathbb{R}$, is said to be conditionally symmetric-invariant if*

$$f(t_x \cdot x) = f(x) \quad , \forall t_x \in S_x, \;\; \forall x \in \Omega,$$

where S_x is a set of transformations unique to element x and $t_x : \Omega \to \Omega$.

It is easy to see that conditionally invariant functions are a generalization of group invariant functions where $S_x \equiv G \;\; \forall x \in \Omega$ where G is a group. The above definition is motivated by the fact that representation for proteins may not necessarily limited to be invariant just to group actions, but to a more general set of protein specific transformations. I detail this motivation next.

A protein molecule is composed of amino acids (say n amino acids), where each atom in the protein belongs to one amino acid $\in \{1, \ldots, n\}$. Excluding the hydrogen atoms, every amino acid contains four atoms known as the backbone atoms (shown in Figure 4.1), and other atoms which belong to the side chain. We shall use m to denote the number of atoms in the protein. Traditionally, protein structures are solved by X-ray crystallography or cyro-EM and the obtained structure is normally considered a unique 3D conformation of the molecule. Molecules available via the Protein Data Bank [123] generally include only unique sets of coordinates to demonstrate the 3D structures, which lead to the unique answer as 'gold standard' in structure prediction, such as protein structure prediction competitions - CASP [124]. Under these assumptions protein side-chain conformations are usually assumed to be clustered into rotamers, which are rigid conformations represented by discrete side-chain dihedral angles.

However, recent works [13], [14] have observed that — while the backbone atoms of all n amino acids in a protein molecule together (i.e. $4n$ atoms) form a rigid structure, its side chains can exhibit very flexible (continuous and discrete) conformations not restricted to just clustered rotamers. The underlying goal of my work is to capture this inherent flexibility of proteins and to ensure different conformers of the same protein get identical representations. The above problem of capturing protein conformations is further compounded by the fact that protein conformations are often times unique to the protein - i.e. conformations exhibited by a protein are input (protein) specific. The desiderata, therefore is a model which outputs conformation invariant representations when the conformations (symmetries) exhibited by a protein varies across proteins when access to only a single protein conformer is available.

I denote a protein as a tuple $p = (V, \boldsymbol{X}_s, \boldsymbol{X}_p)$ (data from pdb files) where V is the set of atoms in the protein, $\boldsymbol{X}_s \in \mathbb{R}^{m \times d}, d \geq 1$ is the scalar atom feature matrix associated with the atoms, $\boldsymbol{X}_p \in \mathbb{R}^{m \times 3}$ are the positional coordinates associated with the atoms in the protein. Without loss of generality, we number the nodes in $V = \{1, ..., n\}$ following the same ordering of the rows in $\boldsymbol{X}_s, \boldsymbol{X}_p$.

Definition 4.1.2 (Rigid Backbone Protein Conformations)**.** *For an m-atom protein p with atom positional coordinates $\boldsymbol{X}_p \in \mathbb{R}^{m\times 3}$ (recall that by definition p contains information about $\boldsymbol{X}_p$), where $C_p \subset \mathbb{R}^{m\times 3}$ denotes the set of viable conformations of p, which keeps the positional coordinates associated with the backbone fixed. I use $T_p \subset \mathbb{R}^{m\times 3\times 3}$ (where a 3x3 matrix is associated with every single atom) to denote the set of non-isometric transformations of p, which yield the set C_p i.e. $\forall c_p \in C_p$, $\exists t_p \in T_p$, s.t. for an atom with index $\mathrm{i} \in \{1, \ldots, m\}$, the new position of atom i is $\boldsymbol{X}_p[\mathrm{i}]t_p[\mathrm{i}]^T = c_p[\mathrm{i}]$, $\boldsymbol{X}_p[\mathrm{i}] \in \mathbb{R}^{1\times 3}, t_p[\mathrm{i}] \in \mathbb{R}^{3\times 3}$.*

T_p forms a set of transformations which acts on the protein atomic coordinates via matrix multiplication. Two elements of T_p (with corresponding matrices of individual atoms and then aggregating for the protein as a whole) may or may not combine via matrix multiplication (as the binary operation) to result in another element of T_p i.e. matrix multiplication (atom corresponding) is not necessarily closed in T_p.

Unfortunately, sampling transformations from T_p to obtain viable protein conformations is an unattainable task, since this would entail first sampling a transformation from $\mathbb{R}^{m\times 3\times 3}$ and then verifying if the transformation is viable (and the vast majority of transformations are not viable). I shall address this issue using an MCMC method - which can then be combined with MCGD training of the neural network [125] to learn conformation invariant protein representations where different viable conformations are obtained via MCMC are used in every batch. But first, I specify the invariance requirements for learning representations of proteins.

A symmetric-invariant function for proteins (per Definition 4.1.1), should be invariant to (i) transformations which change its atomic coordinates to any of its rigid backbone conformations in C_p (ii) transformations which change the atomic coordinates of all its atoms via rigid transformations — group actions such as rotations/ translations (iii) a combination of the two above which transforms it into one of its viable conformations and is then further acted upon by rigid transformations.

4.2 Obtaining Viable Conformations

Before I describe my Markov chain Monte Carlo procedure that will sample atomic coordinates from the set of rigid backbone conformations C_p of protein p (Definition 4.1.2), we need a way to sample from the local neighborhood of a conformation.

4.2.1 Efficiently sampling viable conformations.

To efficiently sample a candidate conformation from C_p, we will follow a multi-step approach: First, we (i) construct a directed tree for every amino acid in the protein to capture the inherent flexibility in protein structures. Then, we (ii) leverage a directed forest of the protein (described next) to transform its atomic coordinates and check for viability.

Part 1. Directed forest construction.

Using the input protein molecule, we construct a directed forest using the following three step procedure:

S1. Each amino acid in the protein gets its own *directed tree*. The atoms in the amino acid's backbone form the base (root) nodes (i.e., there can be multiple base nodes in every amino acid). This is illustrated as node 1 in Figure 4.2's *tree*. The set of base nodes of all the amino acids in the protein are jointly referred to as the base nodes of the protein. We will also refer to them as the base nodes of the directed forest.

S2. The atoms adjacent to the amino acid's backbone via a covalent bond become their immediate children in the *tree*. For example, the node SC1 forms the child of αC in the fig. 4.1.

S3. The other covalent bonds of the children establish the grandchildren of the root, and so on, until all the atoms in the amino acid are a part of the *tree*. For example, SC2 and SC3 become the children of the node SC1 in the directed tree constructed for Figure 4.1.

Figure 4.2 is an illustration of some directed tree constructed with the above procedure, where node 1 is the root node, with nodes $2, 4$ as its children and so on.

It is important to note that, the above procedure ensures that in the constructed *tree*, each (non-root) node has a single parent and may have arbitrarily many children. Further, cycles/rings which are present in the molecule are broken on the basis of bond length, i.e., larger bonds are chosen before smaller bonds. Ties between same-length bonds are broken arbitrarily.

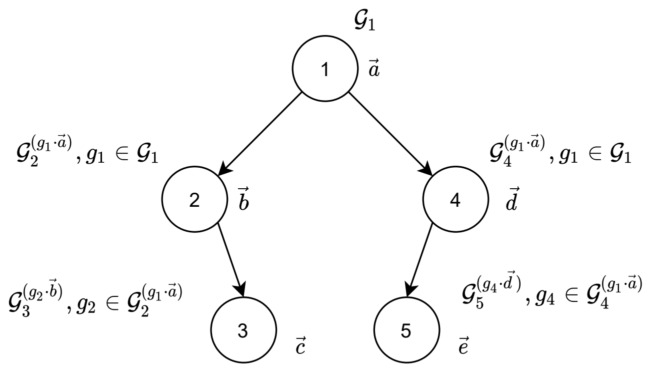

$\mathcal{G}_1$
1
$\vec{a}$
$\mathcal{G}_2^{(g_1\cdot\vec{a})}, g_1 \in \mathcal{G}_1$
$\mathcal{G}_4^{(g_1\cdot\vec{a})}, g_1 \in \mathcal{G}_1$
2
$\vec{b}$
4
$\vec{d}$
$\mathcal{G}_3^{(g_2\cdot\vec{b})}, g_2 \in \mathcal{G}_2^{(g_1\cdot\vec{a})}$
$\mathcal{G}_5^{(g_4\cdot\vec{d})}, g_4 \in \mathcal{G}_4^{(g_1\cdot\vec{a})}$
3
$\vec{c}$
5
$\vec{e}$

Figure 4.2. *Directed tree* corresponding to a set with 5 points which exhibits conditional invariances. Our proposed model, for example, allows node 2 and its descendants to transform its coordinates about node 1 (its parent) upon actions from group $\mathcal{G}_2^{(g_1\cdot\vec{a})}$, $g_1 \in \mathcal{G}_1$. In practice, for protein molecules we use $\mathcal{G}_1 = SO(3)$ and $\mathcal{G}_{\mathrm{i}}^{(\cdot,\cdot)} = SO(3) \forall \mathrm{i} \neq 1$. We note that in protein molecules not all transformations about a node would be allowed due to steric repulsions between atoms as well as potential overlaps of atoms.

Next, we shall look at how the atomic coordinates are transformed using the directed tree.

Part 2. Conditional transformations of atomic coordinates.

Let $\boldsymbol{X}_p$ be the available input atomic coordinates of the protein. To obtain a new candidate protein conformation (say $\boldsymbol{X}_p' \in \mathbb{R}^{m\times 3}$), we sample uniformly, one of the directed trees from the forest.

Next, we transform the atomic coordinates only of the subset of atoms represented in this directed tree. This set is denoted by $A \subset V$. That is, in the candidate conformation, $\boldsymbol{X}_p'[\mathrm{i}] = \boldsymbol{X}_p[\mathrm{i}]$, $\forall \mathrm{i} \in V \setminus A$ — or equivalently $t_p[\mathrm{i}] = \boldsymbol{I}$, $\forall \mathrm{i} \in V \setminus A$.

Starting with the root node, we traverse the directed tree using breadth first search and update all the descendants of the node currently being considered. For the backbone atoms, we leave $\boldsymbol{X}_p[\mathrm{i}] = \boldsymbol{X}'_p[\mathrm{i}]$ (or equivalently $t_p[\mathrm{i}] = I$), i.e., atomic coordinates are unchanged for atoms in the backbone of the amino acid.

Next, we will describe approximate transformations on a given protein through a directed tree using *pointed sets.*

Definition 4.2.1 (Pointed sets)**.** *A pointed set is an ordered pair* (X, x_0)*, where* X *is a set and* $x_0 \in X$ *is called the basepoint.*

For instance, let $M_\mathrm{i} = \{\boldsymbol{X}_p[\mathrm{j}] \; : \; \mathrm{j} \in N_\mathrm{i}\}$, where N_i is the set containing i and all its descendants in its directed tree. Then, $(M_\mathrm{i}, \boldsymbol{X}_p[\mathrm{i}])$ is a pointed set.

Let parent of node i be denoted by $\circ$ and let $G_\mathrm{i}^{(g_\circ \cdot \boldsymbol{X}_p[\circ])}$ (in practice, SO(3)) be the group from which actions $g_\mathrm{i} \in G_\mathrm{i}^{(g_\circ \cdot \boldsymbol{X}_p[\circ])}$ are sampled uniformly — that can transform the positional coordinates of node i and its descendants about its parents in the directed tree, where $g_\circ \cdot \boldsymbol{X}_p[\,\circ\,]$ is the transformed atomic coordinates of the parent of node i (recall, we are using breadth first search - so a top down approach). In the case of the root node (for example 1 in fig. 4.2), the actions are just drawn from G_{root} (or G_1 in fig. 4.2).

The associated transformations of the atomic coordinates of the atoms in N_i can be given by the mapping $h_\mathrm{i} : (M_\mathrm{i}, \boldsymbol{X}_p[\mathrm{i}]) \to (\mathbb{R}^{|N_\mathrm{i}| \times 3}, g_\circ \cdot \boldsymbol{X}_p[\,\circ\,] + g_\mathrm{i} \cdot (\boldsymbol{X}_p[\mathrm{i}] - \boldsymbol{X}_p[\,\circ\,]))$ where the coordinates of node $\mathrm{j} \in N_\mathrm{i}$ are updated as:

$$\boldsymbol{X}'_p[\mathrm{j}] \leftarrow g_\circ \cdot \boldsymbol{X}_p[\,\circ\,] + g_\mathrm{i} \cdot (\boldsymbol{X}_p[\mathrm{j}] - \boldsymbol{X}_p[\,\circ\,]) \tag{4.1}$$

If the node $\mathrm{j} \neq \mathrm{i}$, its coordinates, i.e., $\boldsymbol{X}'_p[\mathrm{j}]$ can be further updated as we traverse the tree. When $G_\mathrm{i}^{(g_\circ \cdot \boldsymbol{X}_p[\circ])} = SO(3)\ \forall \mathrm{i} \in V$, every node in the directed tree can be rotated about its parents as shown in Figure 4.3 - which is the side chain of the amino acid shown in Figure 4.1.

However, each candidate $\boldsymbol{X}'_p$ obtained via the above transformation process may not belong to C_p. To that end protein structural validation tools may be used to check the validity of these candidates. Molprobity [120]–[122] is such a tool,which outputs scores corresponding to different metrics (such as number of dihedral (torsion) angles outside the allowed threshold, number of atom-atom clashes arising due to steric repulsions, etc) which can be compared

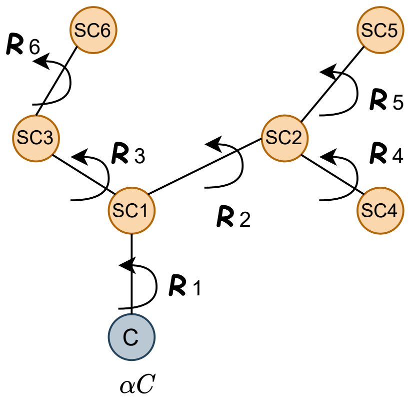


Figure 4.3. Maximum flexibility allowed by our candidate sampling process when the group associated with every node in the tree is SO(3) (the special orthogonal group in 3 dimensions). While not candidate is likely to be accepted such a candidate generation process provides the flexibility for every node to be rotated about its immediate parent while preserving bond lengths.

Algorithm 1 Sampling a viable conformation of c_p

```
1: Input: Conformation $c_p$ of protein $p$
2: Obtain the directed forest corresponding to the protein with $n$ directed trees where $n$ is
   the number of amino acids in the protein.
3: repeat
4:     Sample $y \sim \textsc{unif}(\{1, 2, \cdots, n\})$ and obtain the corresponding directed tree
5:     Traverse through directed tree via BFS and update atomic coordinates via Equation (4.1)
       to obtain $c'_p$ a candidate conformation of $c_p$— where group actions are
       always sampled uniformly from SO(3).
6:     Check if $c'_p$ is a valid protein conformation via protein structure validation tools
7: until $c'_p$ is a valid conformation
8: Return: $c'_p$, a valid conformation of $c_p$
```

with the originally provided conformation $\boldsymbol{X}_p$. Note, that while Molprobity is not perfect and we may end up developing models more invariant than just to viable conformations, alternative (more precise) tools can be employed to check validity. Additionally, the goal of this work is not to use Molprobity/ other protein structure validation tools as a part of generative models and a more invariant model does not necessarily affect performance on unseen but valid protein test data.

Our algorithm to sample viable conformations is summarized in Algorithm 1. The `repeat`-loop proposes a conformation in the *neighborhood* of the current conformation c_p

which is only accepted when the proposed conformation c'_p is a valid conformation. As such it is an acceptance-rejection sampling algorithm. We will see later that the actual sampling probability distribution is not required to be known by our procedure. Our MCMC procedure defined next only requires Algorithm 1 to follow certain conditions which we argue it follows.

4.2.2 Sampling Conformers via MCMC

We now describe the MCMC procedure that, starting at any conformer configuration $c_p^{(0)} \in C_p$ will, in steady state, sample conformations according to a unique stationary distribution which depends on the protein p, where C_p is the set of all viable conformations of p as defined in Definition 4.1.2.

To that end we first define the transition kernel of the Markov chain. Algorithm 1 samples a conformation c'_p in the *neighborhood* of a given conformation c_p using directed a forest. The neighborhood $\mathcal{N}(c_p)$ of the conformation c_p is loosely defined as the set of all possible conformations c'_p which may result from running Algorithm 1 with c_p as input. We denote the probability distribution induced by Algorithm 1 as the transition kernel $\kappa(c'_p \mid c_p)$ which has support $\mathcal{N}(c_p)$.

Definition 4.2.2 (Conformer Sampling Markov Chain (CSMC) $\mathbf{\Phi}_p$)**.** *We define the conformer sampling Markov chain $\mathbf{\Phi}_p$ as a time-homogenous Markov chain over the state space C_p with transition kernel κ as defined above, where C_p is the set of valid conformations associated with given protein p as defined in Definition 4.1.2.*

The consistency of our learning procedure does not depend on the precise definition of κ. However, our procedure relies on the fact that any conformer $c'_p \in C_p$ can be sampled by κ in a finite number of steps, starting from a conformer c_p. We will use this fact later to show that the chain $\mathbf{\Phi}_p$ converges to to a unique stationary distribution regardless of the initial conformer. All proofs are given in the Section C.2.

Proposition 4.2.1. *Given the CSMC $\mathbf{\Phi}_p$ from Definition 4.2.2 whose transitions are governed by κ which is implicitly defined by Algorithm 1 as described above. For any pair of* *conformers $c_p, c'_p \in C_p$, there exists $\tau_p < \infty$, independent of c_p, such that $P^{\tau_p}_{\mathbf{\Phi}_p}(c_p, c'_p) > 0$, where $P^{\tau_p}_{\mathbf{\Phi}_p}$ is the τ_p step transition probability.*

Next, we show the existence of a unique steady state distribution of our Markov chain.

Proposition 4.2.2. *The CSMC $\mathbf{\Phi}_p$ defined in Definition 4.2.2 is uniformly ergodic if Proposition 4.2.1 is satisfied. Specifically there exists a unique steady state distribution $\boldsymbol{\pi}_p$ such that for all $c_p \in C_p$, $\mid P^{n}_{\mathbf{\Phi}_p}(c_p, \cdot) - \boldsymbol{\pi}_p(\cdot) \mid \leq \textsc{c}\,\textsc{r}^n$, where $\textsc{c} < \infty$ and $\textsc{r} < 1$ are constants that depend on $\mathbf{\Phi}_p$, $P^{n}_{\mathbf{\Phi}_p}$ is the n step transition probability and $\mid \cdot \mid$ is the ℓ_1 norm.*

Given that we now have the ability to draw samples from a Markov chain which achieves a unique stationary distribution $\boldsymbol{\pi}_p$ on C_p, we shall leverage this fact, next, in our learning framework to learn conformer invariant representations of proteins.

4.3 Learning Framework

We shall employ a learning strategy, where we use the viable conformations obtained via the Markov chain to learn a function which outputs conformer invariant representations.

Let $\mathcal{D} = \{(x_{\text{j}}, y_{\text{j}})\}_{\text{j}=1}^{N}$ be the input data (where we have a single conformer for every protein in the dataset). We shall consider a single data point $(x_{\text{j}}, y_{\text{j}})$ henceforth, where $x_{\text{j}} = (V, \boldsymbol{X}_s, \boldsymbol{X}_p)$ is the input protein. We consider a supervised learning setting where y_{j} is the associated target. We consider both classification and regression tasks.

Let $C_{\text{j}} = \{c_{\text{j}_\text{i}}\}$ be the set of viable conformations of protein x_{j}. We only consider viable conformations which defer only in the atomic coordinates matrix $\boldsymbol{X}_p$. For a given protein x_{j}, we shall use x_{j_i} to denote protein x_{j} but with $\boldsymbol{X}_p$ of the original protein modified by c_{j_i}, and use $S_{\text{j}} = \{x_{\text{j}_\text{i}}\}$ to denote the set of all viable x_{j_i} i.e. the state space.

Let $f : \Omega \to \mathbb{R}^d$, $d > 0$ be any function with learnable parameters θ^f, (for e.g. any neural network model such as 3D CNN, GNN, Transformer, LSTM, etc.) which takes a protein (in the form $(V, \boldsymbol{X}_s, \boldsymbol{X}_p)$) as input and outputs protein representations. The function f need not necessarily output conformer invariant representations of the protein x_{j}. Then, a simple way to obtain conformer invariant representations of protein x_{j} (apart from using trivial functions such as a constant function or function independent of $\boldsymbol{X}_v$) is computing its expected value over all its viable conformations. We shall denote $\overline{\overline{f}}$ to denote a function which outputs conformer invariant representations of a protein.

$$\overline{\overline{f}}(x_{\mathrm{j}};\theta^{f}) = \mathbb{E}_{X\sim\boldsymbol{\pi}_{\mathrm{j}}}[f(X;\theta^{f})] \tag{4.2}$$

where $\boldsymbol{\pi}_{\mathrm{j}}(\cdot)$ is the steady state distribution of the Markov chain associated with the protein x_{j}. As such, $\overline{\overline{f}}(x_{\mathrm{j}};\theta^{f}) = \overline{\overline{f}}(x'_{\mathrm{j}};\theta^{f})$ for any viable protein conformation $x'_{\mathrm{j}} \in \mathcal{S}_{\mathrm{j}}$. Subsequently, to learn an optimal f (which we denote by $f^{\star}$), we wish to minimize the loss, defined as follows:

$$\begin{aligned} L(\mathcal{D};\theta^{f},\theta^{\rho}) &= \frac{1}{N}\sum_{\mathrm{j}=1}^{N}\hat{L}(y_{\mathrm{i}},\rho(\overline{\overline{f}}(x_{\mathrm{j}};\theta^{f});\theta^{\rho})) \\ &= \lim_{k\to\infty}\frac{1}{N}\sum_{\mathrm{j}=1}^{N}\hat{L}(y_{\mathrm{i}},\rho(\frac{1}{k}\sum_{\mathrm{i}=1}^{k}f(x_{\mathrm{j}}^{\mathrm{i}};\theta^{f});\theta^{\rho})) \end{aligned} \tag{4.3}$$

where $\hat{L}$ is a convex loss function e.g. cross entropy loss, ρ is some differentiable function with parameters θ^{ρ} (in practice, an MLP) and $\lim\limits_{k\to\infty}\frac{1}{k}\sum_{\mathrm{i}=1}^{k}f(x_{\mathrm{j}}^{\mathrm{i}};\theta^{f})$ can be employed as an asymptotically unbiased and consistent estimate of $\overline{\overline{f}}$ where $x_{\mathrm{j}}^{\mathrm{i}}$ is the ith sample from the Markov chain corresponding to the protein x_{j}

Since Equation (4.3) is computationally intractable, we employ a surrogate for the loss given by:

$$J(\mathcal{D};\theta^{f},\theta^{\rho}) = \lim_{k\to\infty}\frac{1}{N}\sum_{\mathrm{j}=1}^{N}\frac{1}{k}\sum_{\mathrm{i}=1}^{k}\hat{L}(y_{\mathrm{i}},\rho(f(x_{\mathrm{j}}^{k};\theta^{f});\theta^{\rho})) \tag{4.4}$$

Observe that in Equation (4.4), the expectation over the conformations is now outside the $\hat{L}$ and ρ functions while still remaining conformation invariant (however, the optimal parameters corresponding to minimizing J are different from miniming L). Following [25], [27], we note that, when ρ is the identity function, Equation (4.4) serves as an upper bound for Equation (4.3) via Jensen's inequality.

Making the case for simplicity, we next show convergence properties using the full batch setting. Note that is procedure and proofs are equally applicable in the mini-batch setting with minor modifications.

Full-batch training: We consider the entire dataset as a single batch $\mathcal{B} = \{(x_1, y_1), \ldots, (x_\mathrm{j}, y_\mathrm{j}), \ldots (x_N$ and for a data point $(x_\mathrm{j}, y_\mathrm{j})$ we denote the corresponding steady state distributions of the corresponding Markov chains using $\boldsymbol{\pi}_\mathrm{j}$ and the corresponding state space as S_j.

We denote all learnable parameters of $\rho \circ f$ as $\theta \equiv (\theta^f, \theta^\rho)$ and consider $\theta \in \Theta \subseteq \mathbb{R}^m$, $m > 0$ such that the function $\rho \circ f$ may be non-convex for some values of θ.

We consider a sequence of step sizes $(\gamma_k)_{k=0}^{\infty}$ which satisfy:

$$\sum_k \gamma_k = +\infty, \quad \sum_k \ln k \cdot \gamma_k^2 < +\infty \tag{4.5}$$

and follow a gradient scheme where parameters are updated as:

$$\theta_k = \theta_{k-1} - \gamma_k \boldsymbol{Z}_k, \quad k > 0 \tag{4.6}$$

where $\boldsymbol{Z}_k = \frac{1}{N} \sum_{\mathrm{j}=1}^{N} \nabla_\theta \hat{L}(y_\mathrm{j}, \rho(f(x_\mathrm{j}^{k-1}; \theta_{k-1}^f); \theta_{k-1}^\rho))$ and x_j^{k-1} is the $k-1^{th}$ sample from the Markov chain corresponding to the data point $(x_\mathrm{j}, y_\mathrm{j})$ and employ the Markov chain gradient descent procedure [125] where the loss in epoch $k, k > 0$ of training is obtained by

$$\hat{J}_k = \frac{1}{N} \sum_{\mathrm{j}=1}^{N} \hat{L}(y_\mathrm{j}, \rho(f(x_\mathrm{j}^{k-1}; \theta_{k-1}^f); \theta_{k-1}^\rho)).$$

and minimizes the surrogate loss given in eq. (4.4) when k is sufficiently large so that the Markov chain has converged to its unique steady state distribution.

Next, we list the set of assumptions we make to ensure convergence of our conformer invariant MCGD procedure.

Assumption 1. *We make the following assumptions:*

1. *For any $\theta \in \Theta$ and $x_\mathrm{j}^k \in S_\mathrm{j}$, the function f is differentiable $\forall \mathrm{j}$*
2. $\sup_{\theta \in \Theta, x_\mathrm{j}^k \in S_\mathrm{j}} \{\| \nabla_\theta \, \rho \circ f(x_\mathrm{j}^k) \|\} < +\infty$ *i.e. the gradients are bounded.*

3. *$\forall x_{\mathrm{j}}^k \in S_{\mathrm{j}}$, $\forall \theta_1, \theta_2 \in \Theta$, $\| \nabla_{\theta_1} \rho \circ f(x_{\mathrm{j}}^k) - \nabla_{\theta_2} \rho \circ f(x_{\mathrm{j}}^k) \| < L \| \theta_1 - \theta_2 \|$ for some $L \geq 0$ i.e., the gradients are $L-$Lipschitz.*
4. *$\mathbb{E}_{x_{\mathrm{j}}^k \sim \pi_{\mathrm{j}}}[\nabla_\theta \, \rho \circ f(x_{\mathrm{j}}^k)] = \nabla_\theta \, \mathbb{E}_{x_{\mathrm{j}}^k \sim \pi_{\mathrm{j}}}[\rho \circ f(x_{\mathrm{j}}^k)]$*

Next, we formally state the convergence of our optimization procedure to optimal parameters $\theta^\star$ which yield conformer invariant protein representations.

Proposition 4.3.1. *Let the step sizes satisfy (4.5) and the function parameters θ be updated as (4.6) and Assumption 1 hold, then the MCGD optimization enjoys properties of almost sure convergence to the optimal θ.*

The use of Markov chain gradient descent procedure to optimize the conformer invariant learning procedure optimizes the objective J in Equation (4.4), and thus has the following implication on how outputs should be calculated at inference time: **Inference time:**

We estimate $\overline{\overline{f}}$ using an empirical average of f evaluated over conformations visited by the Markov chain defined in Definition 4.2.2:

$$\hat{\overline{\overline{f}}}_k(x_{\mathrm{j}}; \theta^f) = \frac{1}{k} \sum_{\mathrm{i}=1}^{k} f(x_{\mathrm{j}}^{(\mathrm{i})}; \theta^f)\,, \tag{4.7}$$

where $x_{\mathrm{j}}^{(\mathrm{i})}$ is the ith state visited by the Markov chain started at x_{j}. Since the Markov chain has a unique steady state distribution, $\hat{\overline{\overline{f}}}_k(x_{\mathrm{j}}; \theta^f)$ is both asymptotically unbiased and consistent. That is $\lim_{k \to \infty} \hat{\overline{\overline{f}}}_k(x_{\mathrm{j}}; \theta^f) \overset{a.s.}{=} \overline{\overline{f}}(x_{\mathrm{j}}; \theta^f)$.

4.4 Related Work

Group Equivariant Neural Networks: Group equivariant neural networks [57], [111], [113], [116], [117], [126]–[128] help capture discrete and continuous symmetries of elements (e.g. images) by introducing group theoretic constraints as inductive biases in the neural network. While group equivariant neural network capture global symmetries, local symmetries of manifold spaces can be captured via gauge equivariant networks [129], [130]. [101], [102], [126] provide a complete review of the theoretical aspects and a wide variet of applications of group equivariant neural networks.

Graph Neural Networks: Graph Neural Networks [2], [7], [9], [65] have gained renewed focus over the past few years and have found applications in recommender systems, biology, chemistry, and many other real world problems which can be formulated as graphs and currently serve as the state of the art in majority of node and graph classification/ regression tasks. Graph neural networks work on the principles of permutation equivariance/ invariances (also groups) and have exploited a message passing framework to learn powerful and expressive representations of nodes/ graphs.

Group Equivariant Graph Neural Networks: These networks combine continuous symmetries (lie groups) with permutation equivariances and has found applications with resounding success on small molecules [114], [115], [118], [119] which exhibit rigid body characteristics. More recently, they have also been applied to learning representations of proteins, which is discussed below.

Neural Networks for Representation Learning of Proteins: Protein representation has gained a lot of attention especially with the tremendous successes of Alphafold and Alphafold2. A variety of neural network architectures including 3D CNNs, LSTM's and Transformers (treating the protein as a sequence) as well as graph neural networks have been employed to exploit the rigid body symmetries of proteins [104], [105], [107], [108], [110], [131]–[133].

Generative Models: Conformation generation models [134]–[140] have also recently gained attend where the goal of the model is to predict 3d structure of molecules given input 2d structure - our objective in this work is completely different, but can be used to improve predictions of the aforementioned models.

Non-Rigid Body Dynamics: Non-Rigid Body Dynamics of objects has long been studied both by physicists and in the fields of computer vision to understand and capture the geometric deformations of objects [141], [142]. To the best of our knowledge, there exists no prior work in deep learning which captures the non rigidity of protein molecules (which cannot be modeled as C^k manifolds).

Unrelated Work with similar names: Non classical and conditional symmetries of solutions to ODE's and PDE's have been discussed in the past - these works while they share a similar title with this work, they have very little in common as we are not dealing with jet spaces or manifolds [143]–[145].

4.5 Results

We evaluate our proposed augmentation procedure on four different tasks from the ATOM3D dataset [146], which we describe next. Results are provided in Table 4.1 and Section 4.5 .

Table 4.1. GVP-GNN - Baseline vs SOTA vs Conformation Invariant Strategies for four different tasks on proteins from the ATOM3D [146] dataset. Corresponding to the metric, ↑ indicates that higher is better, while ↓ indicates that lower is better. Bold values indicate best results for a given row. The values for GVP-GNN were obtained from [107] and for the SOTA from [108], [146]. Gray colored cells indicates that the augmented model outperforms the baseline model.

Task	Metric	Baseline (GVP GNN) [108]	SOTA (Various methods)	Non MCMC Augmented GVP-GNN (Ours)	MCMC Augmented GVP-GNN (Ours)
PSR	Global R_s ↑	0.845 ± 0.004	0.845 ± 0.004	0.806 ± 0.011	**0.852± 0.006**
LEP	AUROC ↑	0.628 ± 0.055	**0.740 ± 0.010**	**0.739 ± 0.060**	0.704 ± 0.039
LBA	RMSE ↓	1.594 ± 0.073	**1.416 ± 0.021**	1.635 ± 0.007	1.435 ± 0.007
MSP	AUROC ↑	0.680 ± 0.015	0.680 ± 0.015	0.799 ± 0.016	**0.857 ± 0.049**

Tasks on Atom3D Datasets

ATOM3D [146] is a unified collection of datasets concerning the 3D structure of proteins. These datasets are specifically designed to provide a benchmark for machine learning methods which operate on 3D molecular structure, and represent a variety of important structural, functional, and engineering tasks. As part of our evaluation, we perform the (a) Protein Structure Ranking (PSR) (b) Protein Mutation Stability Prediction (MSP) (c) Ligand Binding Affinity (LBA) (d) Ligand Efficacy Prediction (LEP) tasks. We describe

Table 4.2. GNN - Baseline vs SOTA vs Conformation Invariant Strategies for four different tasks on proteins from the ATOM3D [146] dataset. Corresponding to the metric, ↑ indicates that higher is better, while ↓ indicates that lower is better. Bold values indicate best results for a given row. The values for GNN were obtained from [146] and for the SOTA from [108], [146]. Gray colored cells indicates that the augmented model outperforms the baseline model.

Task	Metric	Baseline (GNN) [146]	SOTA (Various methods)	Non MCMC Augmented GNN (Ours)	MCMC Augmented GNN (Ours)
PSR	Global R_s ↑	0.755 ± 0.004	**0.845 ± 0.004**	0.766 ± 0.001	0.761 ± 0.004
LEP	AUROC ↑	0.740 ± 0.010	0.740 ± 0.010	0.657 ± 0.008	0.672 ± 0.012
LBA	RMSE ↓	1.570 ± 0.025	**1.416 ± 0.021**	1.520 ± 0.022	1.519 ± 0.022
MSP	AUROC ↑	0.621 ± 0.009	**0.680 ± 0.015**	0.610 ± 0.021	0.662 ± 0.008

each of the four tasks in detail in Section C.3. For all four datasets and tasks we report the same metrics as proposed by the authors.

Model, Baselines and Discussion

We endow the vector gated GVP-GNN (a group invariant graph neural network) model [108] and a GNN using GCN [2] layers with conditional transformations from our proposed MCMC method. It is important to note that when the positions of the atoms are altered the contact graph which are inputs to the base encoders are changed appropriately in every epoch. We also provide a strategy where all the transformations are created from “gold standard” $\boldsymbol{X}_p$ rather than via the MCMC method, i.e., the MCMC is restarted at every epoch during training. We compare our proposed framework against the GVP-GNN model, as well as 3DCNN models [105], [146] - and list the SOTA among the currently available models in Table 4.2. We note that our proposed models outperform the current SOTA in 3 of the 4 tasks and is very close to SOTA performance on the fourth task as well. Moreover, the MCMC method tends to outperform the non MCMC method, which can be attributed to the guarantees it provides to the learning framework.

4.6 Conclusions

This work addresses the limitations of current protein representation learning methods which are unable to learn conformer invariant representations — and hence unable to capture the inherent flexibility present in protein side chains. To address these, we introduced conditional transformations to capture protein structure, while respecting the restrictions posed by constraints on dihedral (torsion) angles and steric repulsions between atoms. Subsequently, we introduced a Markov chain Monte Carlo based framework to learn representations that are invariant to these conditional transformations. Our results have corroborated our proposed strategy on four different protein tasks wherein endowing existing baseline models with these conditional transformations helped improve their performance without sacrificing computational cost.

REFERENCES

[1] A. Grover and J. Leskovec, “Node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2016, pp. 855–864.

[2] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[3] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2014, pp. 701–710.

[4] A. Mnih and R. R. Salakhutdinov, “Probabilistic matrix factorization,” in *Advances in neural information processing systems*, 2008, pp. 1257–1264.

[5] S. Chen, J. L. Moore, D. Turnbull, and T. Joachims, “Playlist prediction via metric embedding,” in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2012, pp. 714–722.

[6] P. Gopalan, F. J. Ruiz, R. Ranganath, and D. Blei, “Bayesian nonparametric poisson factorization for recommendation systems,” in *Artificial Intelligence and Statistics*, 2014, pp. 275–283.

[7] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[8] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.

[9] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” *arXiv preprint arXiv:1810.00826*, 2018.

[10] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, “Weisfeiler and leman go neural: Higher-order graph neural networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4602–4609.

[11] B. Srinivasan and B. Ribeiro, “On the equivalence between positional node embeddings and structural graph representations,” in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SJxzFySKwH.

[12] B. Srinivasan, D. Zheng, and G. Karypis, “Learning over families of sets-hypergraph representation learning for higher order tasks,” in *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, SIAM, 2021, pp. 756–764.

[13] Z. Miao and Y. Cao, “Quantifying side-chain conformational variations in protein structure,” *Scientific reports*, vol. 6, no. 1, pp. 1–10, 2016.

[14] P. Gainza, K. E. Roberts, and B. R. Donald, “Protein design using continuous rotamers,” *PLoS computational biology*, vol. 8, no. 1, e1002335, 2012.

[15] B. Srinivasan, V. Ioannidis, S. Adeshina, M. Kakodkar, G. Karypis, and B. Ribeiro, “Conditional invariances for conformer invariant protein representations,” in *Under Submission*, 2022.

[16] S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski, “A latent variable model approach to pmi-based word embeddings,” *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 385–399, 2016.

[17] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[18] N. C. Stenseth, W. Falck, O. N. Bjørnstad, and C. J. Krebs, “Population regulation in snowshoe hare and canadian lynx: Asymmetric food web configurations between hare and lynx,” *Proceedings of the National Academy of Sciences*, vol. 94, no. 10, pp. 5147–5152, 1997.

[19] M. L. Bates, S. M. B. Nash, D. W. Hawker, J. Norbury, J. S. Stark, and R. A. Cropp, “Construction of a trophically complex near-shore antarctic food web model using the conservative normal framework with structural coexistence,” *Journal of Marine Systems*, vol. 145, pp. 1–14, 2015.

[20] B. Bloem-Reddy and Y. W. Teh, “Probabilistic symmetry and invariant neural networks,” *arXiv preprint arXiv:1901.06082*, 2019.

[21] R. L. Graham and P. M. Winkler, “On isometric embeddings of graphs,” *Transactions of the American mathematical Society*, vol. 288, no. 2, pp. 527–536, 1985.

[22] N. Linial, E. London, and Y. Rabinovich, “The geometry of graphs and some of its algorithmic applications,” *Combinatorica*, vol. 15, no. 2, pp. 215–245, 1995.

[23] D. Galles and J. Pearl, “An axiomatic characterization of causal counterfactuals,” *Foundations of Science*, vol. 3, no. 1, pp. 151–182, 1998.

[24] T. Austin, “On exchangeable random variables and the statistics of large graphs and hypergraphs,” *Probability Surveys*, vol. 5, pp. 80–145, 2008.

[25] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, “Relational pooling for graph representations,” *arXiv preprint arXiv:1903.02541*, 2019.

[26] S. Kamath, A. Orlitsky, D. Pichapati, and A. T. Suresh, "On learning distributions from their samples," in *Conference on Learning Theory*, 2015, pp. 1066–1100.

[27] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs," *arXiv preprint arXiv:1811.01900*, 2018.

[28] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in neural information processing systems*, 2017, pp. 3391–3401.

[29] G. Namata, B. London, L. Getoor, B. Huang, and U. EDU, "Query-driven active surveying for collective classification," in *10th International Workshop on Mining and Learning with Graphs*, 2012, p. 8.

[30] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[31] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[32] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," *ICML*, 2016.

[33] M. Fürer, "On the combinatorial power of the weisfeiler-lehman algorithm," in *International Conference on Algorithms and Complexity*, Springer, 2017, pp. 260–271.

[34] C. Spearman, "" general intelligence," objectively determined and measured," *The American Journal of Psychology*, vol. 15, no. 2, pp. 201–292, 1904.

[35] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in neural information processing systems*, 2002, pp. 585–591.

[36] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM international on conference on information and knowledge management*, ACM, 2015, pp. 891–900.

[37] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*, ACM, 2013, pp. 37–48.

[38] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2016, pp. 1105–1114.

[39] S. Arora, S. Rao, and U. Vazirani, "Expander flows, geometric embeddings and graph partitioning," *Journal of the ACM (JACM)*, vol. 56, no. 2, p. 5, 2009.

[40] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," *arXiv preprint arXiv:1906.04817*, 2019.

[41] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," in *Proceedings of the 2018 World Wide Web Conference*, International World Wide Web Conferences Steering Committee, 2018, pp. 689–698.

[42] D. Tang, D. Liang, T. Jebara, and N. Ruozzi, "Correlated variational auto-encoders," *arXiv preprint arXiv:1905.05335*, 2019.

[43] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," *ICML*, 2019.

[44] I. Abraham, Y. Bartal, and O. Neimany, "Advances in metric embedding theory," in *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, ACM, 2006, pp. 271–286.

[45] J. Bourgain, "On lipschitz embedding of finite metric spaces in hilbert space," *Israel Journal of Mathematics*, vol. 52, no. 1-2, pp. 46–52, 1985.

[46] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, p. 717, 2009.

[47] R. Kleinberg, "Geographic routing using hyperbolic space," in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, IEEE, 2007, pp. 1902–1909.

[48] Y. Rabinovich and R. Raz, "Lower bounds on the distortion of embedding finite metric spaces in graphs," *Discrete & Computational Geometry*, vol. 19, no. 1, pp. 79–94, 1998.

[49] B. Recht, M. Fazel, and P. A. Parrilo, "Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization," *SIAM review*, vol. 52, no. 3, pp. 471–501, 2010.

[50] B. Shaw, B. Huang, and T. Jebara, "Learning a distance metric from a network," in *Advances in Neural Information Processing Systems*, 2011, pp. 1899–1907.

[51] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[52] A. Nematzadeh, S. C. Meylan, and T. L. Griffiths, "Evaluating vector-space models of word representation, or, the unreasonable effectiveness of counting words near other words.," in *CogSci*, 2017.

[53] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li, "Rolx: Structural role extraction & mining in large graphs," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2012, pp. 1231–1239.

[54] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "Struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2017, pp. 385–394.

[55] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, "Learning structural node embeddings via diffusion wavelets," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2018, pp. 1320–1329.

[56] Z. Chen, S. Villar, L. Chen, and J. Bruna, "On the equivalence between graph isomorphism testing and function approximation with gnns," *NeruIPS*, 2019.

[57] R. Kondor and S. Trivedi, "On the generalization of equivariance and convolution in neural networks to the action of compact groups," *ICML*, 2018.

[58] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," *ICML*, 2019.

[59] J. Wood and J. Shawe-Taylor, "A unifying framework for invariant pattern recognition," *Pattern Recognition Letters*, vol. 17, no. 14, pp. 1415–1422, 1996.

[60] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[61] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 1263–1272.

[62] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[63] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*, Springer, 2018, pp. 593–607.

[64] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems*, 2018.

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

[66] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[67] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 3697–3707.

[68] K. Xu, J. Li, M. Zhang, S. S. Du, K.-i. Kawarabayashi, and S. Jegelka, "What can neural networks reason about?" *arXiv preprint arXiv:1905.13211*, 2019.

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

[70] R. A. Rossi, D. Jin, S. Kim, N. K. Ahmed, D. Koutra, and J. B. Lee, "From community to role-based graph embeddings," *arXiv preprint arXiv:1908.08572*, 2019.

[71] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[72] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[73] A. Gammerman, V. Vovk, and V. Vapnik, "Learning by transduction," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 1998, pp. 148–155.

[74] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

[75] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Advances in neural information processing systems*, 2004, pp. 321–328.

[76] R. S. Michalski, "A theory and methodology of inductive learning," in *Machine learning*, Springer, 1983, pp. 83–134.

[77] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of machine learning research*, vol. 7, no. Nov, pp. 2399–2434, 2006.

[78] A. Epasto and B. Perozzi, "Is a single embedding enough? learning node representations that capture multiple social contexts," in *The World Wide Web Conference*, ACM, 2019, pp. 394–404.

[79] P. Goyal, D. Huang, S. R. Chhetri, A. Canedo, J. Shree, and E. Patterson, "Graph representation ensemble learning," *arXiv preprint arXiv:1909.02811*, 2019.

[80] C. Berge, *Hypergraphs: combinatorics of finite sets*. Elsevier, 1984, vol. 45.

[81] S. Agarwal, K. Branson, and S. Belongie, "Higher order learning with graphs," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 17–24.

[82] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Advances in neural information processing systems*, 2007, pp. 1601–1608.

[83] L. Lu and X. Peng, "High-ordered random walks and generalized laplacians on hypergraphs," in *International Workshop on Algorithms and Models for the Web-Graph*, Springer, 2011, pp. 14–25.

[84] A. Bellaachia and M. Al-Dhelaan, "Random walks in hypergraph," in *Proceedings of the 2013 International Conference on Applied Mathematics and Computational Methods, Venice Italy*, 2013, pp. 187–194.

[85] U. Chitra and B. J. Raphael, "Random walks on hypergraphs with edge-dependent vertex weights," *arXiv preprint arXiv:1905.08287*, 2019.

[86] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3558–3565.

[87] S. Bai, F. Zhang, and P. H. Torr, "Hypergraph convolution and hypergraph attention," *arXiv preprint arXiv:1901.08150*, 2019.

[88] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar, "Hypergcn: A new method for training graph convolutional networks on hypergraphs," in *Advances in Neural Information Processing Systems*, 2019, pp. 1509–1520.

[89] R. Zhang, Y. Zou, and J. Ma, "Hyper-sagnn: A self-attention based graph neural network for hypergraphs," *arXiv preprint arXiv:1911.02613*, 2019.

[90] C. Yang, R. Wang, S. Yao, and T. Abdelzaher, “Hypergraph learning with line expansion,” *arXiv preprint arXiv:2005.04843*, 2020.

[91] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[92] L. Babai, P. Erdos, and S. M. Selkow, “Random graph isomorphism,” *SIaM Journal on computing*, vol. 9, no. 3, pp. 628–635, 1980.

[93] E. Wagstaff, F. B. Fuchs, M. Engelcke, I. Posner, and M. Osborne, “On the limitations of representing functions on sets,” *arXiv preprint arXiv:1901.09006*, 2019.

[94] B. Srinivasan, D. Zheng, and G. Karypis, “Learning over families of sets – hypergraph representation learning for higher order tasks,” *CoRR*, vol. abs/2101.07773, 2021. [Online]. Available: https://arxiv.org/abs/2101.07773.

[95] R. Tyshkevich and V. E. Zverovich, “Line hypergraphs,” *Discrete Mathematics*, vol. 161, no. 1-3, pp. 265–283, 1996.

[96] D. J. Aldous, “Representations for partially exchangeable arrays of random variables,” *Journal of Multivariate Analysis*, vol. 11, no. 4, pp. 581–598, 1981.

[97] P. Diaconis, “Finite forms of de finetti’s theorem on exchangeability,” *Synthese*, vol. 36, no. 2, pp. 271–281, 1977.

[98] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with gumbel-softmax,” *arXiv preprint arXiv:1611.01144*, 2016.

[99] C. J. Maddison, A. Mnih, and Y. W. Teh, “The concrete distribution: A continuous relaxation of discrete random variables,” *arXiv preprint arXiv:1611.00712*, 2016.

[100] A. R. Benson, R. Abebe, M. T. Schaub, A. Jadbabaie, and J. Kleinberg, “Simplicial closure and higher-order link prediction,” *Proceedings of the National Academy of Sciences*, vol. 115, no. 48, E11221–E11230, 2018.

[101] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, “Geometric deep learning: Grids, groups, graphs, geodesics, and gauges,” *arXiv preprint arXiv:2104.13478*, 2021.

[102] J. E. Gerken, J. Aronsson, O. Carlsson, H. Linander, F. Ohlsson, C. Petersson, and D. Persson, “Geometric deep learning and equivariant neural networks,” *arXiv preprint arXiv:2105.13926*, 2021.

[103] K. K. Yang, Z. Wu, and F. H. Arnold, “Machine-learning-guided directed evolution for protein engineering,” *Nature methods*, vol. 16, no. 8, pp. 687–694, 2019.

[104] M. Karimi, D. Wu, Z. Wang, and Y. Shen, “Deepaffinity: Interpretable deep learning of compound–protein affinity through unified recurrent and convolutional neural networks,” *Bioinformatics*, vol. 35, no. 18, pp. 3329–3338, 2019.

[105] G. Pagès, B. Charmettant, and S. Grudinin, “Protein model quality assessment using 3d oriented convolutional neural networks,” *Bioinformatics*, vol. 35, no. 18, pp. 3313–3319, 2019.

[106] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[107] B. Jing, S. Eismann, P. Suriana, R. J. Townshend, and R. Dror, “Learning from protein structure with geometric vector perceptrons,” *arXiv preprint arXiv:2009.01411*, 2020.

[108] B. Jing, S. Eismann, P. N. Soni, and R. O. Dror, “Equivariant graph neural networks for 3d macromolecular structure,” *arXiv preprint arXiv:2106.03843*, 2021.

[109] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, p. 1, 2021.

[110] P. Hermosilla, M. Schäfer, M. Lang, G. Fackelmann, P.-P. Vázquez, B. Kozlikova, M. Krone, T. Ritschel, and T. Ropinski, “Intrinsic-extrinsic convolution and pooling for learning on 3d protein structures,” in *International Conference on Learning Representations*, 2021.

[111] T. Cohen and M. Welling, “Group equivariant convolutional networks,” in *International conference on machine learning*, PMLR, 2016, pp. 2990–2999.

[112] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, “Invariant and equivariant graph networks,” *arXiv preprint arXiv:1812.09902*, 2018.

[113] M. Finzi, S. Stanton, P. Izmailov, and A. G. Wilson, “Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data,” in *International Conference on Machine Learning*, PMLR, 2020, pp. 3165–3176.

[114] J. Klicpera, J. Groß, and S. Günnemann, “Directional message passing for molecular graphs,” *arXiv preprint arXiv:2003.03123*, 2020.

[115] B. Anderson, T.-S. Hy, and R. Kondor, “Cormorant: Covariant molecular neural networks,” *arXiv preprint arXiv:1906.04015*, 2019.

[116] F. B. Fuchs, D. E. Worrall, V. Fischer, and M. Welling, "Se (3)-transformers: 3d roto-translation equivariant attention networks," *arXiv preprint arXiv:2006.10503*, 2020.

[117] M. J. Hutchinson, C. Le Lan, S. Zaidi, E. Dupont, Y. W. Teh, and H. Kim, "Lietransformer: Equivariant self-attention for lie groups," in *International Conference on Machine Learning*, PMLR, 2021, pp. 4533–4543.

[118] V. G. Satorras, E. Hoogeboom, and M. Welling, "E (n) equivariant graph neural networks," *arXiv preprint arXiv:2102.09844*, 2021.

[119] S. Batzner, T. E. Smidt, L. Sun, J. P. Mailoa, M. Kornbluth, N. Molinari, and B. Kozinsky, "Se (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials," *arXiv preprint arXiv:2101.03164*, 2021.

[120] V. B. Chen, W. B. Arendall, J. J. Headd, D. A. Keedy, R. M. Immormino, G. J. Kapral, L. W. Murray, J. S. Richardson, and D. C. Richardson, "Molprobity: All-atom structure validation for macromolecular crystallography," *Acta Crystallographica Section D: Biological Crystallography*, vol. 66, no. 1, pp. 12–21, 2010.

[121] I. W. Davis, A. Leaver-Fay, V. B. Chen, J. N. Block, G. J. Kapral, X. Wang, L. W. Murray, W. B. Arendall III, J. Snoeyink, J. S. Richardson, *et al.*, "Molprobity: All-atom contacts and structure validation for proteins and nucleic acids," *Nucleic acids research*, vol. 35, no. suppl_2, W375–W383, 2007.

[122] C. J. Williams, J. J. Headd, N. W. Moriarty, M. G. Prisant, L. L. Videau, L. N. Deis, V. Verma, D. A. Keedy, B. J. Hintze, V. B. Chen, *et al.*, "Molprobity: More and better reference data for improved all-atom structure validation," *Protein Science*, vol. 27, no. 1, pp. 293–315, 2018.

[123] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic acids research*, vol. 28, no. 1, pp. 235–242, 2000.

[124] A. Kryshtafovych, T. Schwede, M. Topf, K. Fidelis, and J. Moult, "Critical assessment of methods of protein structure prediction (casp)—round xiii," *Proteins: Structure, Function, and Bioinformatics*, vol. 87, no. 12, pp. 1011–1020, 2019.

[125] T. Sun, Y. Sun, and W. Yin, "On markov chain gradient descent," *Advances in neural information processing systems*, vol. 31, 2018.

[126] J. E. Lenssen, M. Fey, and P. Libuschewski, "Group equivariant capsule networks," *arXiv preprint arXiv:1806.05086*, 2018.

[127] F. B. Fuchs, E. Wagstaff, J. Dauparas, and I. Posner, “Iterative se (3)-transformers,” *arXiv preprint arXiv:2102.13419*, 2021.

[128] N. Dehmamy, Y. Liu, R. Walters, and R. Yu, “Lie algebra convolutional neural networks with automatic symmetry extraction,” 2020.

[129] T. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling, “Gauge equivariant convolutional networks and the icosahedral cnn,” in *International Conference on Machine Learning*, PMLR, 2019, pp. 1321–1330.

[130] P. De Haan, M. Weiler, T. Cohen, and M. Welling, “Gauge equivariant mesh cnns: Anisotropic convolutions on geometric graphs,” *arXiv preprint arXiv:2003.05425*, 2020.

[131] J. Ingraham, V. K. Garg, R. Barzilay, and T. Jaakkola, “Generative models for graph-based protein design,” 2019.

[132] A. Strokach, D. Becerra, C. Corbi-Verge, A. Perez-Riba, and P. M. Kim, “Fast and flexible protein design using deep graph neural networks,” *Cell Systems*, vol. 11, no. 4, pp. 402–411, 2020.

[133] F. Baldassarre, D. Menéndez Hurtado, A. Elofsson, and H. Azizpour, “Graphqa: Protein model quality assessment using graph convolutional networks,” *Bioinformatics*, vol. 37, no. 3, pp. 360–366, 2021.

[134] E. Mansimov, O. Mahmood, S. Kang, and K. Cho, “Molecular geometry prediction using a deep generative graph neural network,” *Scientific reports*, vol. 9, no. 1, pp. 1–13, 2019.

[135] G. N. Simm, R. Pinsler, G. Csányi, and J. M. Hernández-Lobato, “Symmetry-aware actor-critic for 3d molecular design,” *arXiv preprint arXiv:2011.12747*, 2020.

[136] O. Ganea, L. Pattanaik, C. Coley, R. Barzilay, K. Jensen, W. Green, and T. Jaakkola, “Geomol: Torsional geometric generation of molecular 3d conformer ensembles,” *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[137] M. Xu, W. Wang, S. Luo, C. Shi, Y. Bengio, R. Gomez-Bombarelli, and J. Tang, “An end-to-end framework for molecular conformation generation via bilevel programming,” in *International Conference on Machine Learning*, PMLR, 2021, pp. 11 537–11 547.

[138] M. Xu, S. Luo, Y. Bengio, J. Peng, and J. Tang, “Learning neural generative dynamics for molecular conformation generation,” *arXiv preprint arXiv:2102.10240*, 2021.

[139] C. Shi, S. Luo, M. Xu, and J. Tang, “Learning gradient fields for molecular conformation generation,” in *International Conference on Machine Learning*, PMLR, 2021, pp. 9558–9568.

[140] S. Luo, C. Shi, M. Xu, and J. Tang, "Predicting molecular conformation via dynamic graph score matching," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[141] J. Taylor, A. D. Jepson, and K. N. Kutulakos, "Non-rigid structure from locally-rigid motion," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, 2010, pp. 2761–2768.

[142] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *Proceedings of the IEEE international conference on computer vision workshops*, 2015, pp. 37–45.

[143] A. Joseph, "The theory of conditional invariance. i," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 305, no. 1482, pp. 405–427, 1968.

[144] V. Fushchich and R. Zhdanov, "Conditional symmetry and reduction of partial differential equations," *Ukrainian Mathematical Journal*, vol. 44, no. 7, pp. 875–886, 1992.

[145] P. J. Olver and E. M. Vorob'ev, "Nonclassical and conditional symmetries," *CRC handbook of Lie group analysis of differential equations*, vol. 3, pp. 291–328, 1996.

[146] R. J. Townshend, M. Vögele, P. Suriana, A. Derry, A. Powers, Y. Laloudakis, S. Balachandar, B. Anderson, S. Eismann, R. Kondor, *et al.*, "Atom3d: Tasks on molecules in three dimensions," *arXiv preprint arXiv:2012.04035*, 2020.

[147] O. Kallenberg, *Foundations of modern probability.* Springer Science & Business Media, 2006.

[148] P. K. Gopalan, L. Charlin, and D. Blei, "Content-based recommendations with poisson factorization," in *Advances in Neural Information Processing Systems*, 2014, pp. 3176–3184.

[149] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, 2001, pp. 556–562.

[150] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in neural information processing systems*, 2014, pp. 2177–2185.

[151] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms.* MIT press, 2009.

[152] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[153] S. Fan and B. Huang, “Recurrent collective classification,” *arXiv preprint arXiv:1703.06514*, 2017.

[154] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[155] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin, “Parallel gibbs sampling: From colored fields to thin junction trees,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 324–332.

[156] C. Meng, J. Yang, B. Ribeiro, and J. Neville, “Hats: A hierarchical sequence-attention framework for inductive set-of-sets embeddings,” in *KDD*, 2019.

[157] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[158] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, “Deep graph library: A graph-centric, highly-performant package for graph neural networks,” *arXiv preprint arXiv:1909.01315*, 2019.

[159] S. P. Meyn and R. L. Tweedie, *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

[160] J. Jankauskaitė, B. Jiménez-García, J. Dapkūnas, J. Fernández-Recio, and I. H. Moal, “Skempi 2.0: An updated benchmark of changes in protein–protein binding energy, kinetics and thermodynamics upon mutation,” *Bioinformatics*, vol. 35, no. 3, pp. 462–469, 2019.

[161] R. Wang, X. Fang, Y. Lu, and S. Wang, “The pdbbind database: Collection of binding affinities for protein- ligand complexes with known three-dimensional structures,” *Journal of medicinal chemistry*, vol. 47, no. 12, pp. 2977–2980, 2004.

[162] Z. Liu, Y. Li, L. Han, J. Li, J. Liu, Z. Zhao, W. Nie, Y. Liu, and R. Wang, “Pdb-wide collection of binding data: Current status of the pdbbind database,” *Bioinformatics*, vol. 31, no. 3, pp. 405–412, 2015.

[163] R. A. Friesner, J. L. Banks, R. B. Murphy, T. A. Halgren, J. J. Klicic, D. T. Mainz, M. P. Repasky, E. H. Knoll, M. Shelley, J. K. Perry, *et al.*, “Glide: A new approach for rapid, accurate docking and scoring. 1. method and assessment of docking accuracy,” *Journal of medicinal chemistry*, vol. 47, no. 7, pp. 1739–1749, 2004.

A. APPENDIX TO CHAPTER 2

A.1 Preliminaries

Noise outsourcing, representation learning, and graph representations: The description of our proofs starts with the equivalence between probability models of graphs and graph representations. We start with the concept of noise outsourcing [24, Lemma 3.1] applied to our task —a weaker version of the more general concept of transfer [147, Theorem 6.10] in pushforward measures.

A probability law $\boldsymbol{Z}|(\mathbf{A}, \boldsymbol{X}) \sim p(\cdot|(\mathbf{A}, \boldsymbol{X}))$, $(\mathbf{A}, \boldsymbol{X}) \in \Sigma_n$, can be described [147, Theorem 6.10] by pure random noise $\epsilon \sim \text{Uniform}(0,1)$ independent of $(\mathbf{A}, \boldsymbol{X})$, passing through a deterministic function $\boldsymbol{Z} = f((\mathbf{A}, \boldsymbol{X}), \epsilon)$ —where $f : \Sigma_n \times [0,1] \to \Omega$, where Ω in our task will be a matrix $\Omega = \mathbb{R}^{n \times d}$ defining node representations, $d \geq 1$. That is, the randomness in the conditional $p(z|(\mathbf{A}, \boldsymbol{X}))$ is entirely outsourced to ϵ, as f is deterministic.

Now, consider replacing the graph $G = (\mathbf{A}, \boldsymbol{X})$ by a $\mathcal{G}$-equivariant representation $\Gamma(\mathbf{A}, \boldsymbol{X})$ of its nodes, the output of a neural network $\Gamma : \Sigma_n \to \mathbb{R}^{n \times m}$, $m \geq 1$, that gives an representation to each node in G. If a representation $\Gamma^\star(\mathbf{A}, \boldsymbol{X})$ is such that $\exists f'$ where $\boldsymbol{Z} = f(\mathbf{A}, \boldsymbol{X}, \epsilon) = f'(\Gamma^\star(\mathbf{A}, \boldsymbol{X}), \epsilon)$, $\forall(\mathbf{A}, \boldsymbol{X}) \in \Sigma_n$ and $\forall \epsilon \in [0,1]$, then $\Gamma^\star(\mathbf{A}, \boldsymbol{X})$ does not lose any information when it comes to predicting $\boldsymbol{Z}$. Statistically [147, Proposition 6.13], $\boldsymbol{Z} \perp\!\!\!\perp_{\Gamma^\star(\mathbf{A}, \boldsymbol{X})} (\mathbf{A}, \boldsymbol{X})$. We call $\Gamma^\star(\mathbf{A}, \boldsymbol{X})$ a most-expressive representation of G with respect to $\boldsymbol{Z}$. A most-expressive representation (without qualifications) is one that is most-expressive for any target variable.

Representation learning is powerful precisely because it can learn functions Γ that are compact and can encode most of the information available on the input. And because the most-expressive $\Gamma^\star$ is $\mathcal{G}$-equivariant, it also guarantees that any $\mathcal{G}$-equivariant function over $\Gamma^\star$ that outputs $\boldsymbol{Z}$ is also $\mathcal{G}$-equivariant, without loss of information.

A.2 Proof of Theorems, Lemmas and Corollaries

We restate and prove **Lemma 1**

Lemma 4. *Two nodes $v, u \in V$, have the same most-expressive structural representation $\Gamma^\star(v, \mathbf{A}, \boldsymbol{X}) = \Gamma^\star(u, \mathbf{A}, \boldsymbol{X})$ iff u and v are isomorphic nodes in $G = (\mathbf{A}, \boldsymbol{X})$ (Definition 2.1.6).*

Proof. In this proof, we consider both directions.

($\Rightarrow$) Consider two nodes $v, u \in V$ which satisfy the condition, $\Gamma^\star(v, \mathbf{A}, \boldsymbol{X}) = \Gamma^\star(u, \mathbf{A}, \boldsymbol{X})$ but are not isomorphic in $G = (\mathbf{A}, \boldsymbol{X})$. By contradiction, suppose u and v have different node orbits. This is a contradiction, since the bijective mapping of Definition 2.2.2 would have to take the same input and map them to different outputs.

($\Leftarrow$) By contradiction, consider the two nodes $u, v \in V$ which are isomorphic in $G = (\mathbf{A}, \boldsymbol{X})$ but with different most expressive structural representations i.e. $\Gamma^\star(v, \mathbf{A}, \boldsymbol{X}) \neq \Gamma^\star(u, \mathbf{A}, \boldsymbol{X})$. This is a contradiction, because as per Definition 2.1.6 two nodes should have the same structural representation, which would imply the most expressive structural representation is not a structural representation. Hence, the two nodes should share the same most expressive structural representation. □

Next, we restate and prove Lemma 2

Lemma 5 (Causal modeling through noise outsourcing)**.** *Definition 1 of [23] gives a causal model as a triplet*

$$M = \langle U, V, F \rangle,$$

where U is a set of exogenous variables, V is a set of endogenous variables, and F is a set of functions, such that in the causal model, $v_\mathrm{i} = f(\vec{pa}_\mathrm{i}, u)$ is the realization of random variable $V_\mathrm{i} \in V$ and a sequence of random variables $\vec{PA}_\mathrm{i}$ with $PA_\mathrm{i} \subseteq V \backslash V_\mathrm{i}$ as the endogenous variable parents of variable V_i as given by a directed acyclic graph. Then, there exists a pure random noise ϵ and a set of (measurable) functions $\{g_u\}_{u \in U}$ such that for $V_\mathrm{i} \in V$, V_i can be equivalently defined as $v_\mathrm{i} \overset{a.s.}{=} f(\vec{pa}_\mathrm{i}, g_u(\epsilon_u))$, where ϵ_u has joint distribution $(\epsilon_u)_{\forall u \in U} \overset{a.s.}{=} g'(\epsilon)$ for some Borel measurable function g' and a random variable $\epsilon \sim \text{Uniform}(0,1)$. The latter defines M via noise outsourcing [24].

Proof. By Definition 1 of [23], the set of exogenous variables U are not affected by the set of endogeneous variables V. The noise outsourcing lemma (Lemma 3.1) of [24] (or its more complete version Theorem 6.10 of [147]) shows that any samples of a joint distribution over a set of random variables U can be described as $(u)_{\forall u \in U} \overset{a.s.}{=} g(\epsilon)$, for some Borel measurable function g and a random variable $\epsilon \sim \text{Uniform}(0,1)$. As the composition of two Borel measurable functions is also Borel measurable, it is trivial to show that there exists Borel measurable functions $\{g_u\}_{u \in U}$ and g', such that $u \overset{a.s.}{=} g_u(\epsilon_u)$ and $(\epsilon_u)_{\forall u \in U} \overset{a.s.}{=} g'(\epsilon)$. The latter is trivial since g_u can just be the identity function, $g_u(x) = x$. □

Next, we restate and prove **Lemma 3**

Lemma 6. *The permutation equivariance of p in Definition 2.2.5 implies that, if two proper subsets of nodes $S_1, S_2 \subsetneq V$ are isomorphic, then their marginal node embedding distributions must be the same up to a permutation, i.e., $p((\boldsymbol{Z}_{\mathrm{i}})_{\mathrm{i} \in S_1} | \mathbf{A}, \boldsymbol{X}) = \pi(p((\boldsymbol{Z}_{\mathrm{j}})_{\mathrm{j} \in S_2} | \mathbf{A}, \boldsymbol{X}))$ for some appropriate permutation π.*

Proof. From Definition 2.2.5, it is trivial to observe that two isomorphic nodes $u, v \in V$ in graph $G = (\mathbf{A}, \boldsymbol{X})$ have the same marginal node embedding distributions. In this proof we extend this to node sets $S \subset V$ where $|S| > 1$. We marginalize over $(Z_{\mathrm{i}})_{\mathrm{i} \notin S_1}$ to obtain $p((\boldsymbol{Z}_{\mathrm{i}})_{\mathrm{i} \in S_1} | \mathbf{A}, \boldsymbol{X})$ and in the other case over $(Z_{\mathrm{i}})_{\mathrm{i} \notin S_2}$ to obtain $p((\boldsymbol{Z}_{\mathrm{i}})_{\mathrm{i} \in S_2} | \mathbf{A}, \boldsymbol{X})$ respectively.

We consider 2 cases as follows:

Case 1: $S_1 = S_2$: This is the trivial where S_1 and S_2 are the exact same nodes, hence their marginal distributions are the identical as well by definition.

Case 2: $S_1 \neq S_2$: Since S_1 and S_2 are also given to be isomorphic, it is clear to see that every node in S_1 has an isomorphic equivalent in S_2. In a graph $G = (\mathbf{A}, \boldsymbol{X})$, the above statement conveys that S_2 can be written as a permutation π on S_1, i.e $S_2 = \pi(S_1)$. Now, employing Definition 2.2.5, it is clear to see that $p((\boldsymbol{Z}_{\mathrm{i}})_{\mathrm{i} \in S_1} | \mathbf{A}, \boldsymbol{X}) = \pi(p((\boldsymbol{Z}_{\mathrm{j}})_{\mathrm{j} \in S_2} | \mathbf{A}, \boldsymbol{X}))$ □

Next, we restate and prove **Theorem 2.2.1**

Theorem A.2.1. *Let $\mathcal{S} \subseteq \mathcal{P}(V)$ be a set of subsets of V. Let $\boldsymbol{Y}(\mathcal{S}, \mathbf{A}, \boldsymbol{X}) = (Y(\vec{S}, \mathbf{A}, \boldsymbol{X}))_{S \in \mathcal{S}}$ be a sequence of random variables defined over the sets $S \in \mathcal{S}$ of a graph $G = (\mathbf{A}, \boldsymbol{X})$, such that we define $Y(\vec{S}, \mathbf{A}, \boldsymbol{X}) := Y_{\vec{S}} | \mathbf{A}, \boldsymbol{X}$ and $Y(\vec{S}_1, \mathbf{A}, \boldsymbol{X}) \overset{d}{=} Y(\vec{S}_2, \mathbf{A}, \boldsymbol{X})$ for any two jointly isomorphic subsets $S_1, S_2 \in \mathcal{S}$ in $(\mathbf{A}, \boldsymbol{X})$ (Definition 2.1.7), where $\overset{d}{=}$ means equality in their marginal distributions. Then, there exists a deterministic function φ such that,*

$\boldsymbol{Y}(\mathcal{S}, \mathbf{A}, \boldsymbol{X}) \overset{a.s.}{=} (\varphi(\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X}), \epsilon_S))_{S \in \mathcal{S}}$, where ϵ_S is a pure source of random noise from a joint distribution $p((\epsilon_{S'})_{\forall S' \in \mathcal{S}})$ independent of $\mathbf{A}$ and $\boldsymbol{X}$.

Proof. The case $S = V$ is given in Theorem 12 of [20]. The case $S = \emptyset$ is trivial. The case $S \subsetneq V$, $S \neq \emptyset$, is described as follows with a constructive proof. First consider the case of two isomorphic sets of nodes $S_1, S_2 \in \mathcal{S}$. As by definition $Y(\vec{S}_1, \mathbf{A}, \boldsymbol{X}) \overset{d}{=} Y(\vec{S}_2, \mathbf{A}, \boldsymbol{X})$, we must assume $p(Y_{\vec{S}_1} | \mathbf{A}, \boldsymbol{X}) = p(Y_{\vec{S}_2} | \mathbf{A}, \boldsymbol{X})$. We can now use the transfer theorem [147, Theorem 6.10] to obtain a joint description $Y_{\vec{S}_1} | \mathbf{A}, \boldsymbol{X} \overset{\text{a.s.}}{=} \phi_1(\vec{S}_1, \mathbf{A}, \boldsymbol{X}, \epsilon)$ and $Y_{\vec{S}_2} | \mathbf{A}, \boldsymbol{X} \overset{\text{a.s.}}{=} \phi_2(\vec{S}_2, \mathbf{A}, \boldsymbol{X}, \epsilon)$, where ϵ is a common source of independent noise. As S_1 and S_2 are joint isomorphic (Definition 2.1.7), there exists an isomorphism $S_1 = \text{iso}(\vec{S}_2)$, where $\phi_1(\text{iso}(\vec{S}_2), \mathbf{A}, \boldsymbol{X}, \epsilon) = \phi_1(\vec{S}_1, \mathbf{A}, \boldsymbol{X}, \epsilon)$. Because the distribution given by $\phi_1(\cdot, \epsilon)$ must be isomorphic-invariant in $(\mathbf{A}, \boldsymbol{X})$ and S_1 and S_2 are also isomorphic in $(\mathbf{A}, \mathbf{X})$ then, for all permutation actions $\boldsymbol{\pi} \in \Pi_n$, there exists a new isomorphism iso' such that $\phi_1(\boldsymbol{\pi}(\text{iso}(\vec{S}_2)), \boldsymbol{\pi}(\mathbf{A}), \boldsymbol{\pi}(\boldsymbol{X}), \epsilon) \overset{d}{=} \phi_1(\text{iso}'(\boldsymbol{\pi}(\vec{S}_1)), \boldsymbol{\pi}(\mathbf{A}), \boldsymbol{\pi}(\boldsymbol{X}), \epsilon)$, which allows us to create a function φ' that incorporates iso' into ϕ_1. Due to the isomorphism between S_1 and S_2, we can do the same process for S_2 to arrive at the same function φ'. We can now apply Corollary 6.11 [147] over $(Y_{\vec{S}_1} | \mathbf{A}, \boldsymbol{X}, Y_{\vec{S}_2} | \mathbf{A}, \boldsymbol{X})$ along with a measure-preserving mapping f to show that $Y_{\vec{S}_1} | \mathbf{A}, \boldsymbol{X} \overset{a.s.}{=} \varphi'(\vec{S}_1, \mathbf{A}, \mathbf{X}, \epsilon_1)$ and $Y_{\vec{S}_2} | \mathbf{A}, \boldsymbol{X} \overset{a.s.}{=} \varphi'(\vec{S}_2, \mathbf{A}, \mathbf{X}, \epsilon_2)$, where $(\epsilon_1, \epsilon_2) = f(\epsilon)$. If S_1 and S_2 are not joint isomorphic, we can simply define $\varphi'(S_\text{i}, \cdot) := \phi_\text{i}(S_\text{i}, \cdot)$. Definition 2.2.4 allows us to define a function φ from which we rewrite $\varphi'(\vec{S}_\text{i}, \mathbf{A}, \mathbf{X}, \epsilon_\text{i})$ as $\varphi(\Gamma^\star(\vec{S}_\text{i}, \mathbf{A}, \mathbf{X}), \epsilon_\text{i})$. Applying the same procedure to all $S \in \mathcal{S}$ concludes our proof. □

Next, we restate and prove **Theorem 2.2.2**

Theorem A.2.2 (The statistical equivalence between node embeddings and structural representations)**.** *Let $\boldsymbol{Y}(\mathcal{S}, \mathbf{A}, \boldsymbol{X}) = (Y(\vec{S}, \mathbf{A}, \boldsymbol{X}))_{S \in \mathcal{S}}$ be as in Theorem 2.2.1. Consider a graph $G = (\mathbf{A}, \boldsymbol{X}) \in \Sigma_n$. Let $\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X})$ be a most-expressive structural representation of nodes $S \in \mathcal{S}$ in $(\mathbf{A}, \boldsymbol{X})$. Then,*

$$Y(\vec{S}, \mathbf{A}, \boldsymbol{X}) \perp\!\!\!\perp_{\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X})} \boldsymbol{Z} | \mathbf{A}, \boldsymbol{X}, \quad \forall S \in \mathcal{S},$$

for any node embedding matrix $\boldsymbol{Z}$ that satisfies Definition 2.2.5, where $A \perp\!\!\!\perp_B C$ means A is independent of C given B. Finally, $\forall(\mathbf{A}, \boldsymbol{X}) \in \Sigma_n$, there exists a most-expressive node embedding $\boldsymbol{Z}^\star|\mathbf{A}, \boldsymbol{X}$ such that,

$$\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X}) = \mathbb{E}_{\boldsymbol{Z}^\star}[f^{(|S|)}((\boldsymbol{Z}^\star_v)_{v\in S})|\mathbf{A}, \boldsymbol{X}], \quad \forall S \in \mathcal{S},$$

for some appropriate collection of functions $\{f^{(k)}(\cdot)\}_{k=1,\ldots,n}$.

Proof. In the first part of the proof, for any embedding distribution $p(\boldsymbol{Z}|\mathbf{A}, \boldsymbol{X})$, we note that by Theorem 2.2.1, $y(\vec{S}, \mathbf{A}, \boldsymbol{X}) \overset{a.s.}{=} f'(\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X}), \epsilon_S)$. Hence, $Y(\vec{S}, \mathbf{A}, \boldsymbol{X}) \perp\!\!\!\perp_{\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X})} \boldsymbol{Z}|\mathbf{A}, \boldsymbol{X}$, $\forall S \in \mathcal{S}$, is a direct consequence of Proposition 6.13 in [147].

In the second part of the proof, we construct an orbit over a most-expressive representation of a graph $(\mathbf{A}, \boldsymbol{X})$ of size n, with permutations that act only on unique node ids (node orderings) added as node features: $\Pi'(\mathbf{A}, \boldsymbol{X}) = \{((\Gamma^\star(v, \mathbf{A}, [\boldsymbol{X}, \pi(1, \ldots, n)^T]))_{\forall v\in V}\}_{\forall \pi \in \Pi_n}$, where $[A, b]$ concatenates column vector b as a column of matrix A. Define $\boldsymbol{Z}^\star|\mathbf{A}, \boldsymbol{X}$ as the random variable with a uniform measure over the set $\Pi'(\mathbf{A}, \boldsymbol{X})$. We first prove that $\boldsymbol{Z}^\star|\mathbf{A}, \boldsymbol{X}$ is a most-expressive node embedding. Clearly, $\boldsymbol{Z}^\star|\mathbf{A}, \boldsymbol{X}$ is a node embedding, since the uniform measure over $\Pi'(\mathbf{A}, \boldsymbol{X})$ is $\mathcal{G}$-equivariant. All that is left to show is that we can construct $\Gamma^\star$ of any-size subset $S \in \mathcal{S}$ from $\boldsymbol{Z}^\star|\mathbf{A}, \boldsymbol{X}$ via

$$\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X}) = \mathbb{E}_{\boldsymbol{Z}^\star}[f^{(|S|)}((\boldsymbol{Z}^\star_v)_{v\in S})|\mathbf{A}, \boldsymbol{X}],$$

for some function $f^{(|S|)}$. This part of the proof has a constructive argument and comes in two parts.

Assume $S \in \mathcal{S}$ has no other joint isomorphic set of nodes in $\mathcal{S}$, i.e., $\nexists S_2 \in \mathcal{S}$ such that S and S_2 are joint isomorphic in $(\mathbf{A}, \boldsymbol{X})$. For any such subset of nodes $S \in \mathcal{S}$, and any element $R_\pi \in \Pi'(\mathbf{A}, \boldsymbol{X})$, there is a bijective measurable map between the nodes in S and their positions in the representation vector $R_\pi = (\Gamma^\star(v, \mathbf{A}, [\boldsymbol{X}, \pi(1, \ldots, n)^T]))_{\forall v\in V}$, since all node features are unique and there are no isomorphic nodes under such conditions. Consider the multiset

$$\mathcal{O}_S(\mathbf{A}, \boldsymbol{X}) := \{(\Gamma^\star(v, \mathbf{A}, [\boldsymbol{X}, \pi(1, \ldots, n)^T]))_{\forall v\in S}\}_{\forall \pi \in \Pi_n}$$

of the representations restricted to the set S. We now show that there exists an surjection between $\mathcal{O}_S(\mathbf{A}, \boldsymbol{X})$ and $\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X})$. There is a surjection if for all $S_1, S_2 \in \mathcal{P}^\star(V)$ that are non-isomorphic, it implies $\mathcal{O}_{S_1}(\mathbf{A}, \boldsymbol{X}) \neq \mathcal{O}_{S_2}(\mathbf{A}, \boldsymbol{X})$. The condition is trivial if $|S_1| \neq |S_2|$ as $|\mathcal{O}_{S_1}(\mathbf{A}, \boldsymbol{X})| \neq |\mathcal{O}_{S_2}(\mathbf{A}, \boldsymbol{X})|$. If $|S_1| = |S_2|$, we prove by contradiction. Assume $\mathcal{O}_{S_1}(\mathbf{A}, \boldsymbol{X}) = \mathcal{O}_{S_2}(\mathbf{A}, \boldsymbol{X})$. Because of the unique feature ids and because $\Gamma^\star$ is most-expressive, the representation $\Gamma^\star(v, \mathbf{A}, [\boldsymbol{X}, \pi(1, \ldots, n)^T])$ of node $v \in V$ and permutation $\pi \in \Pi_n$ is unique. As S_1 is not isomorphic to S_2, and both sets have the same size, there must be at least one node $u \in S_1$ that has no isomorphic equivalent in S_2. Hence, there exists $\pi \in \Pi_n$ that gives a unique representation $\Gamma^\star(u, \mathbf{A}, [\boldsymbol{X}, \pi(1, \ldots, n)^T])$ that does not have a matching $\Gamma^\star(v, \mathbf{A}, [\boldsymbol{X}, \pi(1, \ldots, n)^T])$ for any $v \in S_2$ and $\pi' \in \Pi_n$. Therefore, $\exists a \in \mathcal{O}_{S_1}(\mathbf{A}, \boldsymbol{X})$, where $a \notin \mathcal{O}_{S_2}(\mathbf{A}, \boldsymbol{X})$, which is a contradiction since we assumed $\mathcal{O}_{S_1}(\mathbf{A}, \boldsymbol{X}) = \mathcal{O}_{S_2}(\mathbf{A}, \boldsymbol{X})$.

Now that we know there is such a surjection, a possible surjective measurable map between $\mathcal{O}_S(\mathbf{A}, \boldsymbol{X})$ and $\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X})$ is a multiset function that takes $\mathcal{O}_S(\mathbf{A}, \boldsymbol{X})$ and outputs $\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X})$. For finite multisets whose elements are real numbers $\mathbb{R}$, [93] shows that a most-expressive multiset function can be defined as the average of a function $f^{(|S|)}$ over the multiset. The elements of $\mathcal{O}_S(\mathbf{A}, \boldsymbol{X})$ are finite ordered sequences (ordered according to the permutation) and, thus, can be uniquely (bijectively) mapped to the real line with a measurable map, even when $\mathbf{A}$ and $\boldsymbol{X}$ have edge and node attributes defined over the real numbers $\mathbb{R}$. Thus, by [93], there exists some surjective function $f^{(|S|)}$ whose average over $\mathcal{O}_S(\mathbf{A}, \boldsymbol{X})$ give $\Gamma^\star(\vec{S}, \mathbf{A}, \boldsymbol{X})$.

Now assume $S_1, S_2 \subseteq V$ are joint isomorphic in $(\mathbf{A}, \boldsymbol{X})$, $S_1, S_2 \neq \emptyset$. Then, we have concluded that $\mathcal{O}_{S_1}(\mathbf{A}, \boldsymbol{X}) = \mathcal{O}_{S_2}(\mathbf{A}, \boldsymbol{X})$. Fortunately, by Definition 2.2.3, this non-uniqueness is a required property of the structural representations of $\Gamma^\star(S_1, \mathbf{A}, \boldsymbol{X})$ and $\Gamma^\star(S_2, \mathbf{A}, \boldsymbol{X})$, which must satisfy $\Gamma^\star(S_1, \mathbf{A}, \boldsymbol{X}) = \Gamma^\star(S_2, \mathbf{A}, \boldsymbol{X})$ if S_1 and S_2 are joint isomorphic, which concludes our proof. □

Next, we restate and prove **Corollary** 1

Corollary 7. *The node embeddings in Definition 2.2.5 encompass embeddings given by matrix and tensor factorization methods —such as Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), implicit matrix factorization (a.k.a. word2vec)–, latent* *embeddings given by Bayesian graph models —such as Probabilistic Matrix Factorizations (PMFs) and variants—, variational autoencoder methods and graph neural networks that use random lighthouses to extract node embedddings.*

Proof. In Probabilistic Matrix Factorization [4], we have $\mathbf{A}_{uv} \sim \mathcal{N}(Z_u^T Z_v, \sigma_a^2 \boldsymbol{I})$ where $Z_u \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$, $Z_v \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$. We note that the posterior of $P(\boldsymbol{Z}|A)$ is clearly equivariant, satisfying definition 2.2.5, as a permutation action on the nodes requires the same permutation on the $\sigma^2 \boldsymbol{I}$ matrix as well to obtain $\boldsymbol{Z}$. The proof for Poisson Matrix Factorization [6], [148] follows a similar construction to the above, wherein the Normal assumption is replaced by the Poisson distribution.

Moreover, any matrix factorization algorithm gives an equivariant distribution of embeddings if the input matrices are randomly permuted upon input. Specifically, any Singular Value Decomposition (SVD) method satisfies Definition 2.2.5 as the distribution of the eigenvector solutions to degenerate singular values —which are invariant to unitary rotations in the corresponding degenerate eigenspace— will trivially be $\mathcal{G}$-equivariant even if the algorithm itself outputs values dependent on the node ids. Same is true for non-negative matrix factorization [149] and implicit matrix factorization [17], [150].

PGNN's [40] compute the shortest distances between every node of the graph with a predetermined set of ‘anchor’ nodes to encode a distance metric. By definition, using such a distance metric would make the node embeddings learned by this technique $\mathcal{G}$-equivariant. The shortest path between all pairs of nodes in a graph can be seen equivalently as a function of a polynomial in $\mathbf{A}^k$. Alternatively, this can also be represented using the adjacency matrix and computed using the Floyd-Warshall algorithm [151]. The shortest distance is thus a function of $\mathbf{A}$ ignoring the node features $\boldsymbol{X}$. Since the inputs to the GNN comprises of the distance metric, $\mathbf{A}$ and $\boldsymbol{X}$, the node embeddings $\boldsymbol{Z}$ can equivalently seen as a function of $\mathbf{A}$, $\boldsymbol{X}$ and noise. The noise in this case is characterized by the randomized anchor set selection.

In variational auto-encoder models such as CVAE's, GVAE's, Graphite [42], [43], [152] the latent representations $\boldsymbol{Z}$ are learned either via a mean field approximation or are sampled independently of each other i.e. $\boldsymbol{Z} \sim P(\cdot|\mathbf{A}, \boldsymbol{X})$. We note that in the case of the mean field approximation, the probability distribution is a Dirac Delta. It is clear to see that the $\boldsymbol{Z}$

learned in this case is $\mathcal{G}$-equivariant with respect to any permutation action of the nodes in the graph.

□

Next, we restate and prove **Corollary** 2

Corollary 8. *The link prediction task between any two nodes $u, v \in V$ depends only on the most-expressive tuple representation $\Gamma^\star((u,v), \mathbf{A}, \mathbf{X})$. Moreover, $\Gamma^\star((u,v), \mathbf{A}, \mathbf{X})$ always exists for any graph $(\mathbf{A}, \boldsymbol{X})$ and nodes (u,v). Finally, given most-expressive node embeddings $\boldsymbol{Z}^\star$, there exists a function f such that $\Gamma^\star((u,v), \mathbf{A}, \mathbf{X}) = \mathbb{E}_{\boldsymbol{Z}^\star}[f(\boldsymbol{Z}^\star_u, \boldsymbol{Z}^\star_v)]$, $\forall u, v$.*

Proof. It is a consequence of Corollary 3 with $|S| = 2$. □

Next, we restate and prove **Corollary** 3

Corollary 9. *Sample $\boldsymbol{Z}$ according to Definition 2.2.5. Then, we can learn a k-node structural representation of a subset of k nodes $S \subseteq V$, $|S| = k$, simply by learning a function $f^{(k)}$ whose average $\Gamma(\vec{S}, \mathbf{A}, \boldsymbol{X}) = \mathbb{E}[f^{(k)}((Z_v)_{v \in S})]$ can be used to accurately predict $Y(\vec{S}, \mathbf{A}, \boldsymbol{X})$.*

Proof. This proof is a direct application of Theorem 2.2.2 which shows the statistical equivalence between node embeddings and strucutral representations.

Note that $f^{(k)}((\boldsymbol{Z}_v)_{v \in S})$ can equivalently be represented as $f^{(k)}(\varphi(\Gamma(v, \mathbf{A}, \boldsymbol{X})_{v \in S}, \epsilon_S))$ using Theorem 2.2.2 and that the noise ϵ_S is marginalized from the noise distribution of Theorem 2.2.1, still preserving equivariance. With an assumption of the most powerful $f'^{(k)}$, which is able to capture dependencies within the node set [27] and noise ϵ_S, we can replace the above with $f'^{(k)}(\varphi(\Gamma(S, \mathbf{A}, \boldsymbol{X}), \epsilon_S))$ and subsequently compute an expectation over this function to eliminate the noise.

□

Next, we restate and prove **Corollary** 4

Corollary 10. *Transductive and inductive learning are unrelated to the concepts of node embeddings and structural representations.*

Proof. By Theorem 2.2.2, we can build most-expressive any-size joint representations from node embeddings, and we can get node embeddings from any-size most-expressive joint representations. Hence, given enough computational resources, node embeddings and graph representations can have the same generalization performance over any tasks. This shows they are unrelated with the concepts of transductive and inductive learning. □

Next, we restate and prove **Corollary 5**

Corollary 11. *A node embeddings sampling scheme can increase the structural representation power of GNNs.*

Proof. The proof follows as a direct consequence of Theorem 2.2.2, along with [25] which demonstrates RP-GNN as a concrete method to do so. More specifically, appending unique node ids to node features uniformly at random, makes the nodes unique, and can be seen as a strategy to obtain node embeddings which satisfy Definition 2.2.5 using GNN's. By averaging over multiple such node embeddings gives us structural representations more powerful than that of standalone GNN's.

□

A.3 Colliding Graph Neural Networks (CGNNs)

In this section we propose a new variational auto-encoder procedure to obtain node embeddings using neural networks, denoted Colliding Graph Neural Networks (CGNNs). Our sole reason to propose a new auto-encoder method is because we want to test the expressiveness of node embedding auto-encoders —and, unfortunately, existing auto-encoders, such as [43], do not properly account for the dependencies introduced by the colliding variables in the graphical model. In our experiments, shown later, we aggregate multiple node embedding sampled from CGNN to obtain structural representations of the corresponding nodes and node sets.

Node Embedding Auto-encoder.

In CGNN's, we adopt a latent variable approach to learn node embeddings. Corresponding to each evidence feature vector $\boldsymbol{X}_{\mathrm{i},\cdot} \in \mathbb{R}^k \ \forall \ \mathrm{i} \in V$, we introduce a latent variable $\boldsymbol{Z}_{\mathrm{i},\cdot} \in \mathbb{R}^k$. In addition, our graphical model also consists of observed variables $\mathbf{A}_{\mathrm{i},\mathrm{j},\cdot} \in \mathbb{R}^d \ \forall \ \mathrm{i},\mathrm{j} \in V$. These are related through the joint distribution $p(\mathbf{A}, \boldsymbol{X}|\boldsymbol{Z}) = \prod_{\mathrm{i},\mathrm{j} \in V \times V} p(\mathbf{A}_{\mathrm{i},\mathrm{j},\cdot}|\boldsymbol{Z}_{\mathrm{i},\cdot}, \boldsymbol{Z}_{\mathrm{j},\cdot}) \prod_{h \in V} p(\boldsymbol{X}_{h,\cdot}|\boldsymbol{Z}_{h,\cdot})$, which is summarized by the Bayesian network in Figure A.1 in the Appendix. Note that $\mathbf{A}_{\mathrm{i},\mathrm{j},\cdot}$ is a collider, since it is observed and influenced by two hidden variables, $\boldsymbol{Z}_{\mathrm{i},\cdot}, \boldsymbol{Z}_{\mathrm{j},\cdot}$. A neural network is used to learn the joint probability via MCMC, in an unsupervised fashion. The model learns the parameters of the MCMC transition kernel via an unrolled Gibbs Sampler, a templated recurrent model (an MLP with shared weights across Gibbs sampling steps), partially inspired by [153].

The unrolled Gibbs Sampler, starts with a normal distribution of the latent variables $\boldsymbol{Z}_{\mathrm{i},\cdot}^{(0)}$, $\forall \mathrm{i} \in V$, with each $\boldsymbol{Z}_{\mathrm{i},\cdot}^{(0)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ independently, where I is the identity matrix. Subsequently at time steps $t = 1, 2, \ldots$, in accordance with the graphical model, each variable $\boldsymbol{Z}_{\mathrm{i},\cdot}^{(t)}$ is sequentially sampled from its true distribution conditioned on all observed edges of its corresponding node i, in addition to the most-up-to-date latent variables $\boldsymbol{Z}$'s associated with its immediate neighbors. The reparametrization trick [154] allows us to backpropogate through the unrolled Gibbs Sampler. Algorithm 2 in the Appendix details our method. Consequently, this procedure has an effect on the run-time of this technique, which we alleviate by performing Parallel Gibbs Sampling by constructing parallel splashes [155]. Our unsupervised objective is reconstructing the noisy adjacency matrix.

A.4 CGNN Algorithms

The procedure to generate node embeddings used by the CGNN is given by Algorithm 2. Structural representations are computed as an unbiased estimate of the expected value of a function of the node embedding samples as given by Algorithm 3 via a set of sets function [156].

A.5 Further Results

In Table A.1 we provide the results on node classification, link prediction and triad prediction on the Citeseer dataset.

Algorithm 2 Node Embeddings from the Unrolled Gibbs Sampler

```
input  : $\mathbf{A}$, $\boldsymbol{X}$, num-times
output: $\boldsymbol{Z}$
initialization: $\boldsymbol{Z}_u \sim \mathcal{N}(0,1)\ \forall\ u \in V$
while num-times > 0 do
    for $u \in V$ do
        $\forall v \in V$ such that $\mathbf{A}_{uv} = 1$
        hidden $\leftarrow f(\{Z_v\})$; // $f$ is a permutation invariant function
        visible $\leftarrow g(\{X_v\})$; // $g$ is a permutation invariant function
        $Z_u \leftarrow$ MLP (hidden, visible, $X_u$) + Noise // With Reparametrization Trick
        // Equivalently, $\boldsymbol{Z_u} \sim P(\cdot|\{Z_v\},\{X_v\},\{A_{uv}\},X_u)$
    end
    num-times $\leftarrow$ num-times - 1
end
```

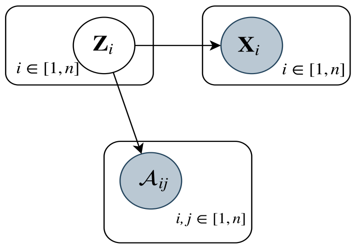


Figure A.1. Latent variable model for Colliding Neural Networks . Observed evidence variables in gray

A.6 Description of Datasets and Experimental Setup

A detailed description of the datasets and the splits are given in Table A.2. Our implementation is in PyTorch using Python 3.6. The implementations for GIN and RP-GIN are done using the PyTorch Geometric Framework. We used two convolutional layers for GIN, RP-GIN since it had the best performance in our tasks (we had tested with 2/3/4/5 convolutional layers). Also since we perform tasks based on node representations rather than graph representations, we ignore the graph wide readout. For GIN and RP-GIN, the embedding dimension was set to 256 at both convolutional layers. All MLPS, across all models have 256 neurons. Optimization is performed with the Adam Optimizer [157]. For the GIN, RP-GIN the learning rate was tuned in {0.01, 0.001, 0.0001, 0.00001} whereas for CGNN's

Algorithm 3 Structural Representations from the Node Embedding Samples

```
input  : $\{\mathbf{Z}^{(\mathrm{i})}\}_{\mathrm{i}=1}^{m}$, $k$ ;// node embedding samples, node set size
output: $g(\boldsymbol{\{}\mathbf{Z}\boldsymbol{\}}_{\boldsymbol{\mathcal{S}}})$; $\mathcal{S} = \{S_1\}_{\forall S_1 \subset V : |S_1| = k}$ //structural representations, $\mathcal{S}$ is a set of sets
initialization: $g(\boldsymbol{\{}\mathbf{Z}\boldsymbol{\}}_{\boldsymbol{\mathcal{S}}}) = \boldsymbol{\{}\vec{0}\boldsymbol{\}}$
for i $\in [1, m]$ do
    for $S \in \mathcal{S}$ do
        $g(\{Z_u\}_{u \in S}) \leftarrow g(\{Z_u\}_{u \in S}) + \frac{1}{m} f(\{Z_u^{(\mathrm{i})}\}_{u \in S})$ // $f$ is a permutation invariant function
    end
end
```

Table A.1. Micro F1 score on three distinct tasks over the Citeseer dataset, averaged over 12 runs with standard deviation in parenthesis. The number within the parenthesis beside the model name indicates the number of Monte Carlo samples used in the estimation of the structural representation. MC-SVD†(1) denotes the SVD procedure run until convergence with one Monte Carlo sample for the representation. Bold values show maximum empirical average, and multiple bolds happen when its standard deviation overlaps with another average.

	Node Classification	Link Prediction	Triad Prediction
Random	0.167	0.500	0.250
GIN(1)	0.701(0.038)	0.543(0.024)	0.309(0.009)
GIN(5)	0.706(0.044)	0.525(0.040)	0.311(0.022)
GIN(20)	0.718(0.034)	0.530(0.023)	0.306(0.012)
RP-GIN(1)	0.719(0.031)	0.541(0.034)	0.313(0.005)
RP-GIN(5)	0.703(0.026)	0.539(0.025)	0.307(0.013)
RP-GIN(20)	0.724(0.020)	0.551(0.030)	0.307(0.017)
1-2-3 GNN(1)	0.189(0.026)	0.499(0.002)	0.306(0.010)
1-2-3 GNN(5)	0.196(0.042)	0.506(0.018)	0.310(0.012)
1-2-3 GNN(20)	0.192(0.029)	0.502(0.014)	0.310(0.020)
MC-SVD†(1)	**0.733(0.007)**	0.552(0.021)	0.304(0.011)
MC-SVD(1)	**0.734(0.007)**	0.562(0.017)	0.297(0.015)
MC-SVD(5)	**0.739(0.006)**	0.556(0.022)	0.302(0.009)
MC-SVD(20)	**0.737(0.005)**	0.565(0.020)	0.299(0.015)
CGNN(1)	0.689(0.010)	0.598(0.024)	0.305(0.009)
CGNN(5)	0.713(0.009)	0.627(0.048)	0.301(0.013)
CGNN(20)	0.721(0.008)	**0.654(0.049)**	0.296(0.008)

the learning rate was set to 0.001. Training was performed on Titan V GPU's. For more details refer to the code provided.

Table A.2. Summary of the datasets

CHARACTERISTIC	**CORA**	**CITESEER**	**PUBMED**	**PPI**
Number of Vertices	2708	3327	19717	56944, 2373[a]
Number of Edges	10556	9104	88648	819994, 41000[a]
Number of Vertex Features	1433	3703	500	50
Number of Classes	7	6	3	121[b]
Number of Training Vertices	1208	1827	18217	44906[c]
Number of Validation Vertices	500	500	500	6514[c]
Number of Test Vertices	1000	1000	1000	5524[c]

[a] The PPI dataset comprises several graphs, so the quantities marked with an "a", represent the average characteristic of all graphs.

[b] For PPI, there are 121 targets, each taking values in $\{0, 1\}$.

[c] All of the training nodes come from 20 graphs while the validation and test nodes come from two graphs each not utilized during training.

B. APPENDIX TO CHAPTER 3

B.1 Additional Examples:

Let $\mathcal{A}$ denote the complete set of substances which are possible components in a prescription drug. Now, given a partial set of substances $X' \subset \mathcal{A}$ part of a single drug, the hyperedge expansion entails completing the set X' as X with a set of substances from $\mathcal{A}$, (which were unobserved due to the data collection procedure for instance), with the set X chosen *s.t.* $X = \arg\max_{X' \subseteq X \subseteq \mathcal{A}} p_{data}(X) - p_{data}(X')$. On the other hand, an example of a hyperedge classification tasks involves determining whether a certain set of substances can form a valid drug or alternatively classifying the nature of a prescription drug. From the above examples, it is clear to see that the hyperedge expansion and hyperedge classification necessitate the framework to jointly capture dependencies between all the elements of an input set (for instance, the associated outputs in these two tasks, requires us to capture all interactions between a set of substances, rather than just the pairwise interactions between a single substances and its neighbors computed independently - as in node classification) and hence are classed as higher order tasks. Additionally, for the hyperedge expansion task, the associated output is a finite set and hence in addition to maximizing the interactions between the constituent elements it is also required to be permutation invariant. For instance, in the expansion task, the training data $X \in \mathcal{X}$ as well as the associated target variable $Y \in \mathcal{Y}$ to be predicted are both sets. The tasks are further compounded by the fact that the training data and the outputs are both relational i.e. the representation of a vertex/ hyperedge also depends on other sets (composition of other observed drugs) in the family of sets i.e. the data is *non i.i.d.*

B.2 Proofs of Properties, Remarks and Theorems

We restate the properties, remark and theorems for convenience and prove them.

Property B.2.1 (Vertex Representations)**.** *The representation of a vertex $v \in V$ in a hypergraph H learnt using Equation (3.2) is a $\mathcal{G}$-invariant representation $\Phi(v, V, E, \boldsymbol{X}, \boldsymbol{E})$ where $\Phi : V \times \Sigma_{n,m} \to \mathbb{R}^d, d \geq 1$ such that*

$$\Phi(v, V, E, \boldsymbol{X}, \boldsymbol{E}) = \Phi((\pi_1(v), \pi_1(V), \pi_2(\pi_1(E)), \pi_1(\boldsymbol{X}), \pi_2(\pi_1(\boldsymbol{E})))$$

$\forall \pi_1 \forall \pi_2$ *where* $\pi_1 \in \mathbb{S}_n$ *and* $\pi_2 \in \mathbb{S}_m$. *Moreover, two vertices v_1, v_2 which belong to the same vertex equivalence class i.e. $v_1 \cong v_2$ obtain the same representation.*

Proof. Part 1: Proof by contradiction. Let $\pi, \pi' \in \mathbb{S}_n$ be two different vertex permutation actions and let $\Phi(\pi(v), \pi(V), E, \pi(\boldsymbol{X}), \boldsymbol{E}) \neq \Phi(\pi'(v), \pi'(V), E, \pi'(\boldsymbol{X}), \boldsymbol{E})$. This implies that the same node gets different representations based on an ordering of the vertex set. From eq. (3.2) it is clear to see that the set function g^k ensures that the vertex representation is not impacted by the edge permutation action. Now let $k = 1$ Expanding eq. (3.2) for both vertex permutation actions and applying the cancellation law of groups, h_v^1 is independent of the permutation action. Since h_v^0 is identical for both, it means the difference arises from the edge permutation action, which is not possible. Now, we can show using induction, the contradiction holds for a certain $k, k \geq 2$, then it holds for $k + 1$ as well. Hence, $\Phi(\pi(v), \pi(V), E, \pi(\boldsymbol{X}), \boldsymbol{E}) = \Phi(\pi'(v), \pi'(V), E, \pi'(\boldsymbol{X}), \boldsymbol{E})$

Part2: Proof by contradiction Let $v_1, v_2' \in V$ be two isomorphic vertices and let $\Phi(v_1, V, E, \boldsymbol{X}, \boldsymbol{E})$ $\neq \Phi(v_2, V, E, \boldsymbol{X}, \boldsymbol{E})$ This implies $h_{v_1}^k \neq h_{v_2}^k \, \forall k \geq 0$ However, by the definition, the two vertices are isomorphic, i.e. they have the same initial node features (if available) i.e. $h_{v_1}^0 = h_{v_2}^0$ and they also posses an isomorphic neighborhood. Equation (3.2) is deterministic, hence the representations obtained by the vertices v_1, v_2 are also identical after 1 iteration i.e. $h_{v_1}^1 = h_{v_2}^1$. Now using induction we can show that, the representations for $h_{v_1}^k$ is the same as $h_{v_2}^k$ for any $k \geq 2$ Hence $\Phi(v_1, V, E, \boldsymbol{X}, \boldsymbol{E}) = \Phi(v_2, V, E, \boldsymbol{X}, \boldsymbol{E})$ when $v_1 \cong v_2$ □

Property B.2.2 (Hyperedge Representations)**.** *The representation of a hyperedge* $\mathrm{e} \in E$ *in a hypergraph H learnt using Equation (3.1) is a $\mathcal{G}$-invariant representation $\Phi(\mathrm{e}, V, E, \boldsymbol{X}, \boldsymbol{E})$ where $\Phi : P^{\star}(V) \times \Sigma_{n,m} \to \mathbb{R}^d, d \geq 1$ such that*

$$\Phi(\mathrm{e}, V, E, \boldsymbol{X}, \boldsymbol{E}) = \Phi((\pi_2(\pi_1(\mathrm{e})), \pi_1(V), \pi_2(\pi_1(E)), \pi_1(\boldsymbol{X}), \pi_2(\pi_1(\boldsymbol{E})))$$

$\forall \pi_1 \forall \pi_2$ where $\pi_1 \in \mathbb{S}_n$ and $\pi_2 \in \mathbb{S}_m$ Moreover, two hyperedges $\mathrm{e}_1, \mathrm{e}_2$ which belong to the same hyperedge equivalence class i.e. $\mathrm{e}_1 \cong \mathrm{e}_2$ obtain the same representation.

Proof. Proof is similar to the two part $\mathcal{G}$-invariant vertex representation proof given above. Replace the vertex permutation action with a joint vertex edge permutation action and similarly use the cancellation law of groups twice. □

Theorem B.2.1 ([95]). *Let H_1, H_2 be hypergraphs without isolated vertices whose line hypergraphs LG_{H_1}, LG_{H_2} are isomorphic. Then $H_1 \cong H_2$ if and only if there exists a bijection $\beta : VLG_{H_1} \rightarrow VLG_{H_2}$ such that $\beta(S_{H_1}) = S_{H_2}$, where S_{H_i} is the family of stars of the hypergraph H_i*

Proof. Theorem is a direct restatement of the theorem in the original work. Please refer to [95] for the proof. □

Theorem B.2.2. *Let H_1, H_2 be two non isomorphic hypergraphs with finite vertex and hyperedge sets and no isolated vertices. If the Weisfeiler-Lehman test of isomorphism decides their line graphs L_{H_1}, L_{H_2} or the star expansions of their duals $H_1^\star, H_2^\star$ to be not isomorphic then there exists a function $\Gamma : \Sigma_{n,m} \rightarrow \mathbb{R}^d$ (via Equation (3.4)) and parameters Θ that maps the hypergraphs H_1, H_2 to different representations.*

Proof. Part 1: Proof by construction, for the line graph L_H. Consider Equation (3.1). By construction, make the set function p as an injective function with a multiplier of a negligible value i.e. $\rightarrow 0$. This implies, a hyperedge only receives information from its adjacent hyperedges. Since we use injective set functions, following the proof of [9] Lemma 2 and Theorem 3, by induction it is easy to see that if the 1-WL isomorphism test decides that the line graphs are non-isomorphic, the representations obtained by the hyperedges through the iterative message passing procedure are also different.

Part 2: Proof by construction, for the dual graph $H^\star$ Again, consider Equation (3.1). By construction, associate a unique identifier with every node and hyperedge in the hypergraph. Construct p as an identity map, this implies, a hyperedge preserves information from which vertices it receives information as well. Since the above p is injective, again following the proof of [9] Lemma 2 and Theorem 3, by induction it is easy to see that if the 1-WL isomorphism test decides that the dual of a hypergraph are non-isomorphic, the representations obtained by the hyperedges through the iterative message passing procedure are also different.

Part 3: From part 1 and part 2 of the proof above, we see that if either the line graphs or the dual of the hypergraphs are distinguishable by the 1-WL isomorphism test as non-isomorphic then our proposed model is able to detect it as well. From the property of vertex representations it also seen that isomorphic vertices obtain the same representation - hence preserving the family of stars representation as well. Now consider Equation (3.4) Now, if the line hypergraphs LG_{H_1} and LG_2 are distinguishable via the line graphs or the dual graphs then the representation obtained by hyperedge aggregations are different. Correspondingly if the family of stars - does not preserve a bijection across the two hypergraphs, then the representation of the graphs are distinguishable using the vertex aggregation. $\square$

Corollary 12. *There exists a function $\Gamma : P^{\star}(V) \times \Sigma_{n,m} \to \mathbb{R}^d$ (via Equation (3.3)) and parameters Θ that map two non-isomorphic hyperedges $\mathrm{e}_1, \mathrm{e}_2$ to different representations.*

Proof. Proof is a direct consequence of the above theorem, eq. (3.3) and above property of hyperedges. $\square$

Remark 4 (Separate Exchangeability)**.** *The representation of a hypergraph H learnt using the function $\Gamma : \Sigma_{n,m} \to \mathbb{R}^d$ (via Equation (3.4)) preserves the separate exchangeability of the incidence structure $\boldsymbol{I}$ of the hypergraph.*

Proof. From Equation (3.4), it is clear that once the representations of the observed vertices and hyperedges are obtained, the vertex permutation actions don't affect the edge permutation and vice versa - i.e. the set functions ϕ, ρ act independently of each other. From Equation (3.4) and through the use of set functions, it is also clear that the representation of the hypergraph is invariant to permutations of both vertex and edge. $\square$

B.3 Description of Datasets and Experimental Setup

In Table B.1 we list the number of vertices and hyperedges for each of the datasets we have considered.

Table B.1. Summary of the datasets

	# Vertices	# Hyperedges
NDC-classes	1161	679
NDC-substances	5556	4916
contact-primary-school	242	4036
contact-high-school	327	1870
threads-math-sx	201863	177398
threads-ask-ubuntu	200974	18785
email-Enron	148	577
email-EU	1005	10631

Our implementation is in PyTorch using Python 3.6. For the hyperedge classification task, we used 5 negative samples for each positive sample. For the hyperedge expansion task, the number of vertices to be added varied from 2 to 7. The implementations for graph neural networks are done using the Deep Graph Library [158]. We used two convolutional layers for all the baselines as well as our model since it had the best performance in our tasks (we had tested with 2/3/4/5 convolutional layers). For all the models, the hidden dimension for the convolutional layers, set functions was tuned from {8,16,32,64}. Optimization is performed with the Adam Optimizer and the learning rate was tuned in {0.1, 0.01, 0.001, 0.0001, 0.00001}. For the set functions we chose from [28] and [27]. For more details refer to the code provided.

C. APPENDIX TO CHAPTER 4

C.1 Group Theory Preliminaries

Definition C.1.1 (Group)**.** *A group is a set G equipped with a binary operation $\cdot : G \times G \to G$ obeying the following axioms:*

- *for all $g_1, g_2 \in G$, $g_1 \cdot g_2 \in G$ (closure).*
- *for all $g_1, g_2, g_3 \in G$, $g_1 \cdot (g_2 \cdot g_3) = (g_1 \cdot g_2) \cdot g_3$ (associativity).*
- *there is a unique $\mathrm{e} \in G$ such that $\mathrm{e} \cdot g = g \cdot \mathrm{e} = g$ for all $g \in G$ (identity).*
- *for all $g \in G$ there exists $g^{-1} \in G$ such that $g \cdot g^{-1} = g^{-1} \cdot g = \mathrm{e}$ (inverse).*

Definition C.1.2 (Group invariant functions)**.** *Let G be a group acting on vector space V. We say that a function $f : V \to \mathbb{R}$ is G-invariant if $f(g \cdot x) = f(x)\ \ \forall x \in V, g \in G$.*

Definition C.1.3 ((Left) Group Action)**.** *For a group G with identity element e, and X is a set, a (left) group action α of G on X is a function $\alpha : G \times X \to X$ that satisfies the following two conditions:*

1. *Identity: $\alpha(\mathrm{e}, x) = x$, $\forall x \in X$*
2. *Compatibility: $\alpha(g, \alpha(h, x)) = \alpha(gh, x)$*

We will use a short hand of $g \cdot x$ for $\alpha(g, x)$ when the action being considered is clear from context.

C.2 Proofs of Propositions

First, we restate and prove Proposition 4.2.1.

Proposition C.2.1. *Given the CSMC $\mathbf{\Phi}_p$ from Definition 4.2.2 whose transitions are governed by κ which is implicitly defined by Algorithm 1 as described above. For any pair of conformers $c_p, c'_p \in C_p$, there exists $\tau_p < \infty$, independent of c_p, such that $P^{\tau_p}_{\mathbf{\Phi}_p}(c_p, c'_p) > 0$, where $P^{\tau_p}_{\mathbf{\Phi}_p}$ is the τ_p step transition probability.*

Proof. Proof by construction. We prove the proposition by showing that one can construct a path $(c_p^{(1)} = c_p, \ldots, c_p^{(t)} = c'_p)$ such that $c_p^{(\mathrm{i})} \in C_p$ and $\kappa(c_p^{(\mathrm{i}+1)} \mid c_p^{(\mathrm{i})}) > 0$, for all $0 < \mathrm{i} < t$, and $t \leq T_p$. The trivial case where $c_p \equiv c'_p$ is proved since every group contains the identity element — sampling the identity element for every node in the directed tree yields the same conformation. Since we consider only non backbone transforming conformations, for the non trivial case, a maximum of $m - 4n$ atoms can differ in positions between any two conformations - where m is the number of atoms in the protein and n is the the number of amino acids in the protein n. Both m, n are finite and we are dealing with continuous conformers (and continuous group actions about every node — groups are closed under their associated binary action, and SO(3) is path connected). So we can traverse between conformers (until we reach the desired conformer) sequentially in a finite number of steps, by using the constructed directed forest - selecting a amino acid (which doesn't violate the viability), fixing the positions of all other amino acids in the protein and rotating the side chain atoms in a single conformer to the final desired state. While this process may result in some side chains being visited multiple times (due to viability constraints), considering continuous conformers and the SO(3) group (which is path connected) ensures we will never reach a state of deadlock. The second condition is satisfied because every group action has an inverse and we only use transformations from SO(3) for every node in the directed trees. $\square$

Next, we restate and and prove proposition 4.2.2

Proposition C.2.2. *The CSMC $\mathbf{\Phi}_p$ defined in Definition 4.2.2 is uniformly ergodic if Proposition 4.2.1 is satisfied. Specifically there exists a unique steady state distribution $\boldsymbol{\pi}_p$ such that for all $c_p \in C_p$,*

$mid P^n_{\mathbf{\Phi}_p}(c_p, \cdot) - \boldsymbol{\pi}_p(\cdot)$

$mid \leq \text{C}\,\text{R}^n$, where $\text{C} < \infty$ and $\text{R} < 1$ are constants that depend on $\mathbf{\Phi}_p$, $P^n_{\mathbf{\Phi}_p}$ is the n step transition probability and

$mid \cdot$

mid is the ℓ_1 norm.

Proof. By Proposition 4.2.1, $\mathbf{\Phi}_p$ satisfies Doeblin's condition as defined in page 396 of Meyn and Tweedie [159] which states that for $c_p, c'_p \in C_p$, $P^{T_p}_{\mathbf{\Phi}_p}(c_p, c'_p) > \epsilon$ for some $\epsilon > 0$ [1]. The

[1]↑We note that this is a simplified version of the actual statement which is defined on the σ-algebra over C_p denoted by $\sigma(C_p)$. Our proof holds when $c'_p \in \sigma(C_p)$

uniform ergodicity then holds due to Theorems 16.2.3 and 16.2.1 from Meyn and Tweedie [159]. □

Next, we restate and prove Proposition 4.3.1. We also restate the required assumptions for the proposition.

Assumption 2. *We make the following assumptions:*

1. *For any $\theta \in \Theta$ and $x_{\mathrm{j}}^{k} \in S_{\mathrm{j}}$, the function f is differentiable* $\forall \mathrm{j}$
2. $\sup_{\theta\in\Theta, x_{\mathrm{j}}^{k}\in S_{\mathrm{j}}}\{|| \nabla_\theta\ \rho \circ f(x_{\mathrm{j}}^{k})\ ||\} < +\infty$ *i.e. the gradients are bounded.*
3. $\forall x_{\mathrm{j}}^{k} \in S_{\mathrm{j}}$, $\forall \theta_1, \theta_2 \in \Theta$, $|| \nabla_{\theta_1}\ \rho \circ f(x_{\mathrm{j}}^{k}) - \nabla_{\theta_2}\ \rho \circ f(x_{\mathrm{j}}^{k})\ || < L\ ||\ \theta_1 - \theta_2\ ||$ *for some* $L \geq 0$ *i.e., the gradients are* $L-$*Lipschitz.*
4. $\mathbb{E}_{x_{\mathrm{j}}^{k}\sim\pi_{\mathrm{j}}}[\nabla_\theta\ \rho \circ f(x_{\mathrm{j}}^{k})] = \nabla_\theta\ \mathbb{E}_{x_{\mathrm{j}}^{k}\sim\pi_{\mathrm{j}}}[\rho \circ f(x_{\mathrm{j}}^{k})]$

Proposition C.2.3. *Let the step sizes satisfy (4.5) and the function parameters θ be updated as (4.6) and Assumption 1 hold, then the MCGD optimization enjoys properties of almost sure convergence to the optimal θ.*

Proof. Given that each protein has an associated time homogeneous Markov Chain with a unique steady state, independent of other proteins, the set of proteins in a mini-batch also form a Markov chain with a unique steady state. We then leverage Corollary 2 (Page 12) of Sun, Sun, and Yin [125] along with Proposition 4.3.1 to ensure almost sure convergence to the optimal θ. □

C.3 Details about datasets and tasks

PSR: This task utilizes data from the structural models submitted to the Critical Assessment of Structure Prediction competition (CASP - [124] - a blind protein structure prediction competition) to rank protein structures from the experimentally determined structure of the protein. The problem is formulated as a regression task, where we predict the global distance test of each structural model from the experimentally determined structure. As prescribed by the dataset authors, the dataset is split by competition years.

MSP: The goal of this task is to identify mutations that stabilize a protein's interactions which forms an important step towards the design of new proteins. This task is significant as probing mutations experimentally techniques are labor-intensive. Atom3D [146] derives this dataset by collecting single-point mutations from the SKEMPI database [160] and model each mutation into the structure to produce a mutated structure. The learning problem is then formulated as a binary classification task where the goal is to predict whether the stability of the complex increases as a result of the mutation. We employ the same splits as suggested by the dataset authors wherein the protein complexes are split such that no protein in the test dataset has more than 30% sequence identity with any protein in the training dataset.

LBA: This task deals with the problem of predicting the strength (affinity) of a candidate drug molecule's interaction with a target protein. The dataset is constructed using the PDBBind database [161], [162], a curated database containing protein-ligand complexes from the PDB and their corresponding binding strengths (affinities). The task is formulated as a regression task with the goal to predict $pK = -\log_{10}(K)$, where K is the binding affinity in Molar units. The splits are created such that no protein in the test dataset has more than 30% sequence identity with any protein in the training dataset.

LEP: The shape of protein impacts whether a protein is in an on or off state which plays an important role in predicting the shape a protein will favor during drug design.This dataset is obtained by curating proteins from several families with both “active” and “inactive” state structures, and model in 527 small molecules with known activating or inactivating function using the program Glide [163]. The task is formulated as a binary classification task where the goal is to predict whether a molecule bound to the structures will be an activator of the protein's function or not. We use the same split as recommended by the ATOM3D authors.

Table C.1. Summary of the datasets

Task	# Train	# Val	# Test	Original Source
MSP	2864	937	247	SKEMPI [160]
LBA	3563	448	452	PDBBlind [161]
LEP	304	110	104	PDB [123]
PSR	25400	2800	16099	CASP [124]