

THE UNIVERSITY OF CHICAGO

GRAPH REPRESENTATION LEARNING, DEEP GENERATIVE MODELS ON
GRAPHS, GROUP EQUIVARIANT MOLECULAR NEURAL NETWORKS AND
MULTIRESOLUTION MACHINE LEARNING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY

HY TRUONG SON

CHICAGO, ILLINOIS

JUNE 2022

Copyright © 2022 by Hy Truong Son
All Rights Reserved

Luận văn Tiến sĩ này xin được dành tặng ông nội Hy Văn Sử, bà nội Công Thị Gái, ông ngoại Công Phuong Châu, bà ngoại Công Thị Bé, bố Hy Văn Thắng, mẹ Công Thị Thu Thủy, chị gái Hy Thị Hồng Nhung và tất cả mọi người thân tôi yêu quý.

This PhD thesis is dedicated to my paternal grandfather Hy Van Su, my paternal grandmother Cong Thi Gai, my maternal grandfather Cong Phuong Chau, my maternal grandmother Cong Thi Be, my father Hy Van Thang, my mother Cong Thi Thu Thuy, my older sister Hy Thi Hong Nhung and all my loved ones.

TABLE OF CONTENTS

LIST OF FIGURES	ix
LIST OF TABLES	xiii
ACKNOWLEDGMENTS	xiii
ABSTRACT	xiv
1 THESIS INTRODUCTION	1
1.1 Graph representation learning	1
1.1.1 Graph formalism	3
1.1.2 Learning tasks on graphs	7
1.1.3 Machine learning methods on graphs	9
1.2 Deep generative models on graphs	11
1.2.1 Traditional approaches	12
1.2.2 Data-driven approaches	14
1.2.3 Evaluating graph generation	23
1.3 Group equivariant molecular neural networks	24
1.3.1 Symmetry-preserving neural networks	24
1.3.2 Rotational equivariant neural networks	28
1.4 Multiresolution Machine Learning	39
2 THESIS OVERVIEW	44
2.1 General structure	44
2.2 Publications	46
2.3 Softwares	47
3 GRAPH REPRESENTATION LEARNING	49
3.1 Chapter Introduction	49
3.2 Graph Kernels	50
3.2.1 Weisfeiler-Lehman graph kernel	52
3.2.2 Optimal assignment kernel and histogram-alignment WL graph feature	54
3.2.3 Dictionary WL graph feature	56
3.2.4 Shortest path graph kernel	61
3.2.5 Graphlet kernel	63
3.2.6 Random walk graph kernel	64
3.2.7 Laplacian graph kernels	66
3.2.8 Diffusion kernels	69
3.3 Using graph kernels	72
3.3.1 Gaussian Processes	72
3.3.2 Kernelized ridge regression	76
3.3.3 Support Vector Machines	78

3.3.4	Kernel principal component analysis	86
3.4	Message Passing Neural Networks and its variants	89
3.4.1	Label propagation algorithm	89
3.4.2	Generic graph neural network	90
3.4.3	Neural graph fingerprint	91
3.4.4	Learning convolutional neural networks for graphs	94
3.4.5	Gated graph neural networks	96
3.4.6	Weisfeiler-Lehman network	97
3.4.7	Message Passing Neural Networks	99
3.4.8	Interaction networks	100
3.4.9	Molecular graph convolutions	100
3.4.10	Deep tensor neural networks	101
3.5	Learning graphs on the spectral domain	102
3.6	Graph-based Semi-Supervised Learning	103
3.7	Covariant Compositional Networks	104
3.7.1	Compositional networks	106
3.7.2	Permutation equivariant networks	110
3.7.3	Covariant aggregation functions	116
3.7.4	Architecture	122
3.7.5	Learning to estimate Density Functional Theory calculation	129
3.8	GraphFlow Deep Learning Framework	130
3.8.1	Motivation for a new deep learning framework	130
3.8.2	Philosophy of the design	131
3.8.3	Parallelization	133
3.8.4	Performance evaluation	138
3.9	Experiments	140
3.10	Software	146
3.11	Chapter Conclusion	148
4	MULTIRESOLUTION EQUIVARIANT GRAPH VARIATIONAL AUTOENCODER	150
4.1	Chapter Introduction	150
4.2	Related work	151
4.3	Multiresolution graph network	152
4.3.1	Construction	152
4.3.2	Higher order message passing	155
4.3.3	Learning to cluster	158
4.4	Hierarchical generative model	160
4.4.1	Background on graph variational autoencoder	160
4.4.2	Multiresolution VAEs	162
4.5	Markov Random Fields	164
4.6	Equivariant learnable prior	166
4.7	Experiments	167
4.7.1	Molecular graph generation	167

4.7.2	Unsupervised molecular properties prediction on QM9	176
4.7.3	Supervised molecular properties prediction on ZINC	177
4.7.4	General graph generation by MGVAE	179
4.7.5	Link prediction on citation graphs by MGVAE	180
4.7.6	Graph-based image generation by MGVAE	183
4.8	Software	185
4.9	Chapter Conclusion	187
5	ROTATIONALLY EQUIVARIANT MOLECULAR NEURAL NETWORKS . . .	190
5.1	Chapter Introduction	190
5.2	Related work	190
5.3	Motivation from physics	192
5.3.1	The multipole expansion	192
5.3.2	Spherical tensors	194
5.4	Molecular neural networks based on the Clebsch-Gordan layer	195
5.5	Experiments	197
5.5.1	Learning the ground state properties	197
5.5.2	Learning potential energy surfaces	198
5.6	Software	200
5.7	Chapter Conclusion	200
6	LEARNING MULTIRESOLUTION MATRIX FACTORIZATION AND ITS WAVELET NETWORKS ON GRAPHS	202
6.1	Chapter Introduction	202
6.2	Related work	203
6.3	Notation	204
6.4	Multiresolution Matrix Factorization	206
6.4.1	Background	206
6.4.2	Multiresolution analysis	208
6.4.3	Optimization	211
6.4.4	Proposal of a learning algorithm	212
6.5	Stiefel Manifold Optimization	213
6.6	Reinforcement Learning	219
6.6.1	Problem formulation	219
6.6.2	Markov Decision Process	219
6.6.3	Policy gradient methods	222
6.6.4	Graph convolutional policy network	223
6.6.5	The learning algorithm	225
6.6.6	2-phase process & Transfer learning	225
6.7	Wavelet Networks on Graphs	227
6.7.1	Motivation	227
6.7.2	Network construction	228
6.8	Experiments	230
6.8.1	Matrix factorization	230

6.8.2	Node classification on citation graphs	232
6.8.3	Graph classification	233
6.9	Software	234
6.10	Chapter Conclusion	235
7	FUTURE WORKS	236
7.1	Symmetry-preserving and equivariant models with higher order interactions: more theoretical understanding	236
7.2	Large-scale implementation of the generalized N-Body networks	237
7.2.1	Application in LHC particle tracking & particle discovery	238
7.3	Learning and generating hierarchical multiscale structures and highly sym- metric structures by deep generative models	239
7.3.1	Modeling and generating proteins	239
7.3.2	Classifying and generating crystalized structures	240
7.3.3	Molecule optimization by deep generative models	241
7.4	Multiresolution tensor factorization and hierarchical data visualization	241
8	THESIS CONCLUSION	242
	REFERENCES	243
9	APPENDIX: GROUP & REPRESENTATION THEORY	271
9.1	Chapter Introduction	271
9.2	Groups	271
9.2.1	Basic definitions	271
9.2.2	Subgroups & Cosets	272
9.3	Representations	274
9.3.1	Matrix groups & Character	274
9.3.2	Reducible & Irreducible representations	276
9.4	Fourier Transforms On Groups	280
9.4.1	Classical Fourier Transform	280
9.4.2	Generalized Fourier transform on finite groups	288
9.4.3	Fast Fourier Transform for group $(\mathbb{Z}/2\mathbb{Z})^k$	290
9.4.4	Group theoretic interpretation of the FFT	292
9.4.5	Generalized FFTs	295
9.4.6	Quantum Fourier Transform	296
10	APPENDIX: WAVELET THEORY	304
10.1	Chapter Introduction	304
10.2	Classical Wavelet Transform	307
10.3	Spectral Graph Theory	308
10.3.1	Graph Laplacian	309
10.3.2	Graph Fourier Transform	310
10.4	Spectral Graph Wavelet Transform	311

10.4.1	Construction	311
10.4.2	Main result	312
10.5	Diffusion wavelets	314
10.5.1	Diffusion operators	314
10.5.2	Algorithm & Discussion	315

LIST OF FIGURES

1.1 Visualization of the <i>Laniakea Supercluster</i> that is the home to the <i>Milky Way</i> , our galaxy, and approximately 100,000 other nearby galaxies. The name <i>Laniākea</i> means “immense Heaven” in Hawaiian. The figure depicts the intensity of shock waves in the cosmic gas (blue) around collapsed dark matter structures (orange/white). Credit: TNG Collaboration.	4
1.2 Graphs appear at every scale in Nature, from gigantic galaxies to the quantum world of tiny molecules and particles. The left figure shows the molecular graph of Aspirin, also known as acetylsalicylic acid, with chemical formula C ₉ H ₈ O ₄ . The molecule contains in total 21 atoms including 9 Carbon, 8 Hydrogen and 4 Oxygen; and 21 bonds. However, for simplicity, the molecular graph only considers heavy atoms as nodes, thus it has 13 nodes and 13 edges. The right figure illustrates three-dimensional ball-and-stick model of the Aspirin molecule, as found in the solid state.	4
1.3 Graph variational autoencoder for molecule generation.	17
1.4 Demonstration of the definition of a G -invariant function in the right. The output is unchanged with respect to transformation of the input (see the simplified diagram in the left).	27
1.5 Demonstration of the definition of a G -equivariant or G -covariant function. The output is transformed according to the transformation of the input (see the simplified diagram in the left).	27
3.1 (a) A small graph G with 6 vertices and its adjacency matrix. (b) An alternative form G' of the same graph, derived from G by renumbering the vertices by a permutation $\sigma: \{1, 2, \dots, 6\} \mapsto \{1, 2, \dots, 6\}$. The adjacency matrices of G and G' are different, but topologically they represent the same graph. Therefore, we expect the feature map ϕ to satisfy $\phi(G) = \phi(G')$	105
3.2 Left: Molecular graphs for C ₁₈ H ₉ N ₃ OSSe and C ₂₂ H ₁₅ NSeSi from the Harvard Clean Energy Project (HCEP) dataset with corresponding adjacency matrices. Center and right: The comp-net of a graph \mathcal{G} is constructed by decomposing \mathcal{G} into a hierarchy of subgraphs $\{\mathcal{P}_i\}$ and forming a neural network \mathcal{N} in which each “neuron” n_i corresponds to one of the \mathcal{P}_i subgraphs, and receives inputs from other neurons that correspond to smaller subgraphs contained in \mathcal{P}_i . The center pane shows how this can equivalently be thought of as an algorithm in which each vertex of \mathcal{G} receives and aggregates messages from its neighbors. To keep the figure simple, we only marked aggregation at a single vertex in each round (layer).	108

3.3	In a convolutional neural network if the input image is translated by some amount (t_1, t_2) , what used to fall in the receptive field of neuron $\mathbf{n}_{i,j}^\ell$ is moved to the receptive field of $\mathbf{n}_{i+t_1,j+t_2}^\ell$. Therefore, the activations transform in the very simple way $f'_{i+t_1,j+t_2}^\ell = f_{i,j}^\ell$. In contrast, rotations not only move the receptive fields around, but also permute the neurons in the receptive field internally, so the activations transform in a more complicated manner. The right hand figure shows that if the CNN has a horizontal filter (blue) and a vertical one (red), then after a rotation by 90 degrees, their activations are exchanged. In steerable CNNs, if (i, j) is moved to (i', j') by the transformation, then $f'_{i',j'}^\ell = R(f_{i,j}^\ell)$, where R is some fixed linear function of the rotation angle.	111
3.4	These two graphs are not isomorphic, but after a single round of message passing (red arrows), the labels (activations) at vertices 2 and 3 will be identical in both graphs. Therefore, in the second round of message passing vertex 1 will get the same messages in both graphs (blue arrows), and will have no way of distinguishing whether 5 and 5' are the same vertex or not.	112
3.5	Feature tensors in a first order CCN for ethylene (C_2H_4) assuming three channels (red, green, and blue). Vertices e_1, e_2, e_5, e_6 are hydrogen atoms, while vertices e_3, e_4 are carbons. Edge (e_3, e_4) is a double bond, the other edges are single bonds. (a) At the input layer, the receptive field for each atom is just the atom itself, so the feature matrix has size 1×3 . (b) At level $\ell = 1$ the size of the receptive field of each atom grows depending on the local topology. The receptive fields for each hydrogen atoms grows to include its neighboring carbon atom, resulting in a feature tensor of size 2×3 ; the receptive fields of the carbon atoms grow to include four atoms each, and therefore have size 4×3 . (c) At layer $\ell = 2$, the receptive fields of the carbons will include every atom in the molecule, while the receptive fields of the hydrogens will only be of size 4.	116
3.6	Schematic of the aggregation process at a given vertex v of \mathcal{G} in a second order CCN not involving tensor products with $A \downarrow_{\mathcal{P}_b}$ and assuming a single input channel. Feature tensors F_0, \dots, F_3 are collected from the neighbors of v as well as from v itself, promoted, and stacked to form a third order tensor T . In this example we compute just three reductions Q_1, Q_2, Q_3 . These are then combined them with the $w_{c,j}$ weights and passed through the σ nonlinearity to form the output tensors $(F^{(1)}, F^{(2)}, F^{(3)})$. For simplicity, in this figure the “ ℓ ” and “ a ” indices are suppressed.	118
3.7	Stacking second order tensors (matrices) $A^{(k)}$ s into a third order tensor C	119
3.8	Hadamard/element-wise product preserves covariance.	119
3.9	Tensor product between a first order tensor (vector) B with a second order tensor (local adjacency matrix) A results into a third order tensor C	120
3.10	There are six different ways of covariantly reducing (contracting) a third order tensor A to a second order tensor (matrix) C : by either projecting into one side or taking the diagonal.	121
3.11	Neural activations of the second order CCN on C_2H_4 molecular graph: (a) initialization, (b) after 1 iteration, and (c) after 2 iterations.	128

3.12	Zeroth, first and second order message passing	128
3.13	GraphFlow overview	133
3.14	GPU implementations of tensor contractions in CCN 2D with the Virtual Indexing System. In this figure, d denotes the size of the graph and c denotes the number of channels.	135
3.15	CPU multi-threading for gradient computation	137
3.16	Example of data flow between GPU and main memory	138
3.17	GPU vs CPU tensor contraction running time (milliseconds) in \log_{10} scale	139
3.18	2D PCA projections of Weisfeiler-Lehman features in HCEP	144
3.19	2D PCA projections of CCNs graph representations in HCEP	144
3.20	2D t-SNE projections of Weisfeiler-Lehman features in HCEP	145
3.21	2D t-SNE projections of CCNs graph representations in HCEP	145
3.22	Molecular graph of C_2H_4 (left) and its corresponding line graph (right).	146
3.23	Distributions of ground-truth and prediction of CCN 1D & 2D in HCEP	148
4.1	Aspirin $C_9H_8O_4$, its 3-cluster partition and the corresponding coarsen graph	152
4.2	Hierarchy of 3-level Multiresolution Graph Network on Aspirin molecular graph	155
4.3	MGVAE generates molecules on QM9 (4 on the left) and ZINC (the rest) equivariantly. Both equivariant MGVAE and autoregressive MGN generate high-quality molecules with complicated structures such as rings.	168
4.4	Some generated examples on QM9 by the all-at-once MGVAE with second order \mathbb{S}_n -equivariant decoders.	171
4.5	Some generated examples on QM9 by the all-at-once MGVAE with a MLP decoder instead of the second order \mathbb{S}_n -equivariant one. It generates more tree-like structures.	172
4.6	Some generated examples on QM9 by the autoregressive MGN.	172
4.7	Some generated examples on ZINC by the all-at-once MGVAE with second order \mathbb{S}_n -equivariant decoders. In addition of graph features such as one-hot atomic types, we include several chemical features computed from RDKit (as in Table 4.3) as the input for the encoders. A generated example can contain more than one connected components, each of them is a valid molecule.	173
4.8	Some generated molecules on ZINC by the autoregressive MGN with high QED (drug-likeness score).	174
4.9	Interpolation on the latent space: we randomly select two molecules from ZINC and we reconstruct the corresponding molecular graphs on the interpolation line between the two latents.	175
4.10	The top row includes generated examples and the bottom row includes training examples on the synthetic 2-community dataset.	180
4.11	EGO-SMALL	181
4.12	An image of digit 8 from MNIST (left) and its grid graph representation at 16×16 resolution level (right).	186
4.13	An example of reconstruction on each resolution level for a test image in MNIST.	187
4.14	Generated examples at the highest 32×32 resolution level.	188
4.15	Generated examples at the 16×16 resolution level.	189

5.1	Learning on molecular data specified by a set of charge-position pairs (Z_i, \mathbf{r}_i) for each atom.	191
5.2	Potential expansion.	193
5.3	Effects of a 90° CCW-rotation on the input charges and corresponding moments. .	194
6.1	Multiresolution analysis splits each function space $\mathbb{V}_0, \mathbb{V}_1, \dots$ into the direct sum of a smoother part $\mathbb{V}_{\ell+1}$ and a rougher part $\mathbb{W}_{\ell+1}$	211
6.2	A rotation matrix of order k . The purpose of permutation matrix Π is solely to ensure that the blocks of the matrices appear contiguous in the figure. In this case, $n = 17$ and $k = 4$	211
6.3	MMF can be thought of as a process of successively compressing \mathbf{A} to size $d_1 \times d_1$, $d_2 \times d_2$, etc. (plus the diagonal entries) down to the final $d_L \times d_L$ core-diagonal matrix \mathbf{H} (see Def. 6.4.3). The role of permutation matrix Π is purely for the ease of visualization (as in Fig. 6.2).	212
6.4	Matrix approximation as in Eq. 6.5. In this figure, the core block size of each rotation matrix \mathbf{U}_ℓ and \mathbf{H} are $k \times k = 4 \times 4$ and $d_L \times d_L = 8 \times 8$, respectively. Permutation matrix Π is only for visualization (as in Figs. 6.2 6.3).	212
6.5	Matrix factorization for the Karate network (left), Kronecker matrix (middle), and Cayley tree (right). Our learnable MMF consistently outperforms the classic greed methods.	231
6.6	Visualization of some of the wavelets on the Cayley tree of 46 vertices. The low index wavelets (low ℓ) are highly localized, whereas the high index ones are smoother and spread out over large parts of the graph.	231
9.1	The left is the DCT basis functions used for JPEG image compression, and the right is the basis functions of the inverse DCT. My source code is available at https://github.com/HyTruongSon/Fourier-Transform-Library	285
9.2	My picture and its 2D Fast Fourier Transform.	288
9.3	Efficient circuit derived from the product representation from Eq. 9.8 for the QFT. Swap gates at the end of the circuit which reverse the order of the qubits, and normalization factor of $1/\sqrt{2}$ of the outputs are not shown. At most $n/2$ swaps are needed at the end. This figure is taken from (Nielsen and Chuang, 2010).303	
10.1	University of Chicago's coat of arms and its Daubechies wavelet transform. The most commonly used set of discrete wavelet transforms was proposed by (Daubechies, 1988) in which the formulation is based on the use of recurrence relations to generate progressively finer discrete samplings of an implicit mother wavelet function; each resolution is twice of the previous scale. Remark: The coat of arms was resized to the size of 512×512 and uncolored as input to the wavelet transform. .	307
10.2	Diagram for downsampling, orthogonalization and operator compression (Coifman and Maggioni, 2006).	316

LIST OF TABLES

3.1	GPU vs CPU tensor contraction running time (milliseconds)	139
3.2	GPU and CPU network evaluation running time (milliseconds)	140
3.3	Single thread vs Multiple threads running time	141
3.4	HCEP regression results	146
3.5	QM9(a) regression results (MAE)	147
3.6	QM9(b) regression results (MAE)	147
4.1	Molecular graph generation results. GraphVAE results are taken from (Liu et al., 2018).	169
4.2	All-at-once MGVAE with MLP decoder vs. second order decoder.	171
4.3	The list of chemical/atomic features used for the all-at-once MGVAE on ZINC. We denote each feature by its API in RDKit.	171
4.4	SMILES of the generated molecules included in Fig. 4.8. Online drawing tool: https://pubchem.ncbi.nlm.nih.gov/edit3/index.html	175
4.5	Description and statistics of 13 learning targets on QM9.	176
4.6	Unsupervised molecular representation learning by MGVAE to predict molecular properties calculated by DFT on QM9 dataset.	178
4.7	Supervised MGN to predict solubility on ZINC dataset.	179
4.8	Graph generation results depicting MMD for various graph statistics between the test set and generated graphs. MGVAE outperforms all competing methods.	180
4.9	Citation graph link prediction results (AUC & AP)	182
4.10	Learning to cluster algorithm returns balanced cuts on Cora.	184
4.11	Learning to cluster algorithm returns balanced cuts on Citeseer.	184
4.12	Quantitative evaluation of the generated set by FID metric for each resolution level on MNIST. It is important to note that the generation for each resolution is done separately: for the ℓ -th resolution, we sample a random vector of size $d_z = 256$ from $\mathcal{N}(0, 1)$, and use the global decoder $\mathbf{d}^{(\ell)}$ to decode into the corresponding image size. The baselines are taken from (Dieng et al., 2019).	186
5.1	Mean absolute error of various prediction targets on QM-9. Results within 5% of the best model are indicated in bold.	199
5.2	Mean absolute error of conformational energies (in units of kcal/mol) on MD-17. Results within 5% of the best model are indicated in bold.	200
6.1	Node classification on citation graphs. Baseline results are taken from (Xu et al., 2019).	233
6.2	Graph classification. Baseline results are taken from (Maron et al., 2019b).	234

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Professor Risi Kondor – who has supported me, helped me, guided me, and conveyed continually and convincingly to all of us, his students, a spirit of discovery in regard to research and scholarship. I really appreciate the feedback and suggestions from my PhD committee members Professor Eric Jonas and Professor Yuxin Chen, and also my MSc committee members Professor Michael Maire and Professor Sanjay Krishnan. My most sincere gratitude for Professor András Lőrincz – who has nurtured the beginning of my scientific career during my undergraduate study in Budapest, Hungary; and friends and colleagues at the Department of Informatics at Eötvös Loránd University. I would like to thank Dr. Brandon Anderson and Dr. Erik Thiede, who have been postdoctoral researchers in Risi’s group, for our fruitful collaboration, meaningful discussion and their valuable support. I would like to thank one of my friends, Dr. Adam Dziedzik, for his comments to improve my thesis defense; and my friends and colleagues, Nguyen Duc Hai, Chris Jones and Horace Pan, for our cooperation and discussion, and for their contribution to some ideas and figures in this thesis. A very special thanks to the entire Machine Learning group and the Department of Computer Science at the University of Chicago where has been my home for the last six years.

Finally, this thesis is dedicated to my family who are always by my side. Your love and sacrifices are the most beautiful of all.

ABSTRACT

Graph neural networks (GNNs) utilizing various ways of generalizing the concept of convolution to graphs have been widely applied to many learning tasks, including modeling physical systems, finding molecular representations to estimate quantum chemical computation, etc. Most existing GNNs address permutation invariance by conceiving of the network as a message passing scheme, where each node sums the feature vectors coming from its neighbors. We argue that this scheme imposes a limitation on the representation power of GNNs such that each node loses their identity after being aggregated by summing. Thus, we propose a new general architecture called *Covariant Compositional Networks* (CCNs) in which the node features are represented by higher order tensors and transform covariantly/equivariantly according to a specific representation of the symmetry group of its receptive field. Experiments show that CCNs can outperform competing methods on standard graph learning benchmarks and on estimating the molecular properties calculated by computationally expensive Density Functional Theory (DFT). This novel machine learning approach allows scientists to efficiently extract chemical knowledge and explore the increasingly growing chemical data.

Understanding graphs in a multiscale perspective is essential for capturing the large-scale structure of molecules, proteins, genomes, etc. For this reason, we introduce *Multiresolution Equivariant Graph Variational Autoencoder* (MGVAE), the first hierarchical generative model to learn and generate graphs in a multiresolution and equivariant manner. MGVAE is built upon *Multiresolution Graph Network* (MGN), an architecture which explicitly learns a multilevel hard clustering of the vertices, leading to a true multiresolution hierarchy. MGVAE then employs the hierarchical variational autoencoder model to stochastically generate a graph in multiple resolution levels given the hierarchy of latent distributions. Our proposed framework achieves competitive results with several generative tasks including general graph generation, molecule generation, unsupervised molecular representation learning, link

prediction on citation graphs, and graph-based image generation. Future applications of MGVAE range from lead optimization enhancing the most promising compounds in drug discovery to finding stable crystal structures in material science.

In general, we want to learn on molecular data specified by a set of charge-position pairs for each atom. This problem is invariant to rotations and translations. We use covariant activations to “bake-in” these symmetries, while retaining local geometric information. We propose *Covariant Molecular Neural Networks* (Cormorant), a rotationally covariant neural network architecture for learning the behavior and properties of complex many-body physical systems. We apply these networks to molecular systems with two goals: learning atomic potential energy surfaces for use in Molecular Dynamics simulations, and learning ground state properties of molecules calculated by Density Functional Theory. Some of the key features of our network are that (a) each neuron explicitly corresponds to a subset of atoms; (b) the activation of each neuron is covariant to rotations, ensuring that overall the network is fully rotationally invariant. Furthermore, the non-linearity in our network is based upon tensor products and the Clebsch-Gordan decomposition, allowing the network to operate entirely in Fourier space. Cormorant significantly outperforms competing algorithms in learning molecular Potential Energy Surfaces from conformational geometries in the MD-17 dataset, and is competitive with other methods at learning geometric, energetic, electronic, and thermodynamic properties of molecules on the GDB-9 dataset.

Multiresolution Matrix Factorization (MMF) is unusual amongst fast matrix factorization algorithms in that it does not make a low rank assumption. This makes MMF especially well suited to modeling certain types of graphs with complex multiscale or hierarchical structure. While MMF promises to yield a useful wavelet basis, finding the factorization itself is hard, and existing greedy methods tend to be brittle. Therefore, we propose a “learnable” ver-

sion of MMF that carefully optimizes the factorization with a combination of Reinforcement Learning and Stiefel manifold optimization through back-propagating errors. Based on the wavelet basis produced by MMF when factorizing the normalized graph Laplacian, a wavelet network learning graphs on the spectral domain is constructed with the graph convolution defined via the sparse wavelet transform. We have shown that the wavelet basis resulted from our learnable MMF far outperforms prior MMF algorithms, and the corresponding wavelet networks yield state of the art results on standard node classification on citation graphs and molecular graph classification. This is a promising direction to understand and visualize complex hierarchical structures such as social networks and biological data.

Keywords

Graph neural networks, message passing, permutation equivariance, rotational equivariance, group equivariant neural networks, molecular representation, hierarchical generative models, graph generation, matrix factorization, reinforcement learning, manifold optimization, multiresolution analysis, graph wavelets, wavelet neural networks.

CHAPTER 1

THESIS INTRODUCTION

1.1 Graph representation learning

It has been apparent that graph is one of the most fundamental ways humans organize knowledge and information. In mathematics, graphs are mathematical structures used to model pairwise relations/interactions/connections between objects. A graph in this context is simply a collection of nodes, also called vertices, which are connected by edges, also called links. A large number of interesting structures and phenomena of the world can be described by graphs. For example, with scale ranking from the largest to the smallest in space:

1. It might be possible to say that the whole Universe is a single, possibly infinite network, where the edges are events or interactions between elementary particles, and the nodes are the particles themselves. This is a graph with perhaps 10^{80} nodes (see Figure 1.1). Recently, cosmological simulations have been playing an essential role in discovering mysteries of the Universe, including those of dark matter and dark energy. Perhaps analyzing the graph-theoretical structure of the cosmological graph with the help of advanced Machine Learning and high-performance computation can accelerate the computationally expensive cosmological simulations of a universe that evolves over billions upon billions of years, and consequentially enhance our understanding of the global structure of the Universe in reality.
2. The Internet is the global system of interconnected computer networks using the standardized communication protocols, which facilitates various information and communication systems such as the World Wide Web. The fast-growing Internet also gives rise to many other networks: the network of hyperlinks, distributed data bases, internet-based social networks, etc. Knowledge of the graph-theoretical structure of the Internet

can be exploited for attaining efficiency and comprehensiveness in Web navigation, for designing communication protocols, for analyzing how fast computer viruses spread, etc.

3. The acquaintance graph of all people on Earth forms the largest social network that is the fundamental object of many studies in economics, sociology, history and epidemiology, etc. The internet-based social networks are indeed subgraphs of this gigantic all-people acquaintance graph. It is important to acquire knowledge from these social networks via graph-based mathematical methods to make prediction on how quick news, inventions, technologies, and diseases spread over the world; and to determine the social and public policies accordingly.
4. The human brain has been known to be one of the most complex networks, that consists of about 10^{11} neurons connected by about 10^{14} synapses. Indeed, one of the greatest scientific challenges is to understand ourselves. In the field of Machine Learning and Deep Learning, artificial neural networks, that are analogous to the networks of neurons and are designed to emulate how the brain performs certain computations, have given computers the ability to identify patterns in large, complex datasets and have achieved remarkable successes in the past decade. The growing interaction and mutual reinforcement between artificial intelligence and cognitive science might enable neuroscientists to get further insights into how computation works in the brain, and allow computer scientists and engineers to design algorithms and machines that can have more human-like intelligence in the future.
5. A transistor is an electronic, three-terminal semiconductor device that controls the movement of electrons, amplifies or switches the current flow. Nowadays, there can be more than a billion transistors on a chip that form some of the largest networks in engineering. Despite the fact that these networks are completely man-made and

thoroughly designed, many of their properties such as the exact time they need to perform a certain computation, are hard to determine from their blueprint due to the large size. There arises the need for data-driven methods powered by machine learning and graph algorithms to automatically generate new chip designs that are comparable or superior to those created by humans in all key metrics, including power consumption, performance and chip area; but in a fraction of the time it would take humans to produce an equivalent result.

6. In mathematical chemistry, a molecular graph is a representation of the structural formula of a chemical compound. In terms of graph theory, the vertices and edges of a molecular graph represent the atoms and bonds of the corresponding molecule (see Figure 1.2). In recent years, molecular datasets containing reliable quantum-mechanical properties for millions of molecules are becoming increasingly available. Meanwhile, the process of chemical discovery to find molecules including drugs, antivirals, antibiotics, and battery materials, etc. with desired attributes is a time-consuming and expensive process. Therefore, it is necessary to develop novel machine learning tools and advanced graph learning algorithms to extract chemical knowledge from these datasets and efficiently explore the increasingly growing chemical spaces.

To understand the behavior of these systems as a whole, one has to study the behavior of the individual elements, numerous local substructures as well as the global structure of the underlying network.

1.1.1 Graph formalism

Formally, a graph $G = (V, E)$ is defined by a set of nodes V and a set of edges E between these nodes. A convenient way to represent *simple graphs*, where there is at most one edge

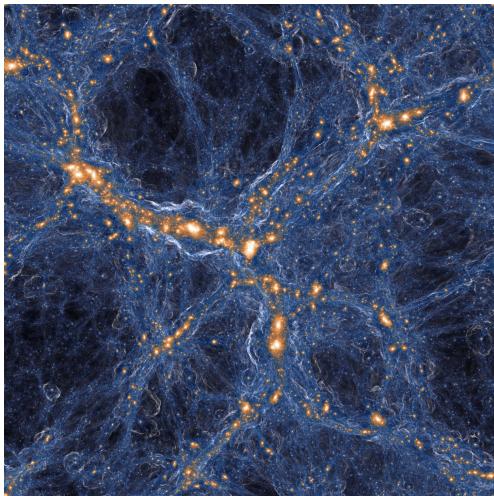


Figure 1.1: Visualization of the *Laniakea Supercluster* that is the home to the *Milky Way*, our galaxy, and approximately 100,000 other nearby galaxies. The name *Laniākea* means “immense Heaven” in Hawaiian. The figure depicts the intensity of shock waves in the cosmic gas (blue) around collapsed dark matter structures (orange/white). Credit: TNG Collaboration.

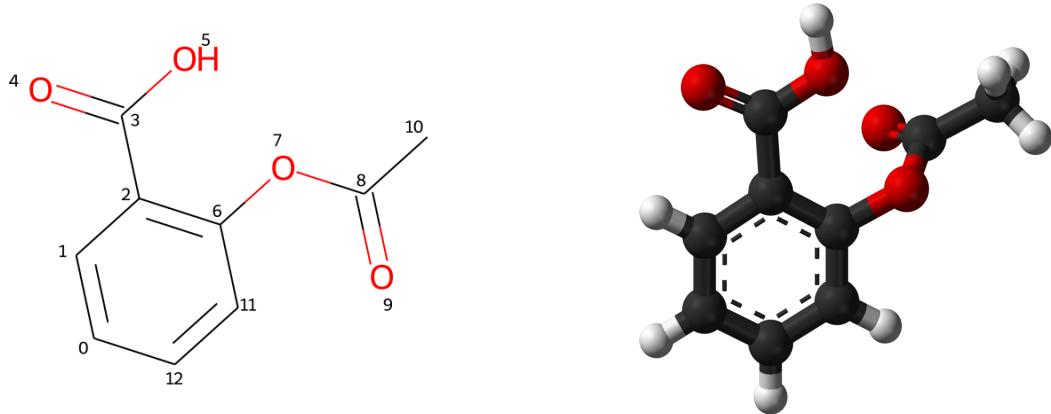


Figure 1.2: Graphs appear at every scale in Nature, from gigantic galaxies to the quantum world of tiny molecules and particles. The left figure shows the molecular graph of Aspirin, also known as acetylsalicylic acid, with chemical formula $C_9H_8O_4$. The molecule contains in total 21 atoms including 9 Carbon, 8 Hydrogen and 4 Oxygen; and 21 bonds. However, for simplicity, the molecular graph only considers heavy atoms as nodes, thus it has 13 nodes and 13 edges. The right figure illustrates three-dimensional ball-and-stick model of the Aspirin molecule, as found in the solid state.

between each pair of nodes, is through an *adjacency matrix* $A \in \mathbb{R}^{|V| \times |V|}$:

$$A_{v_1, v_2} = \begin{cases} 1 & \text{if } (v_1, v_2) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

If the graph is *undirected* (i.e. containing only undirected edges) then A will be a symmetric matrix, but if the graph is *directed* then A will not necessarily be symmetric. If the graph is *weighted* (i.e. containing weighted edges) then the entries of the adjacency matrix are arbitrary values rather than discrete $\{0, 1\}$. For example, a protein-protein interaction graph is a weighted graph in which each node represents a protein and each edge's weight indicates the strength of the association between two corresponding proteins. The multi-relational graphs are the ones having each edge associated with a relation type, e.g., $(v_1, r, v_2) \in E$ denotes an edge of relation type r connecting two nodes v_1 and v_2 . Per edge type r , we can define one adjacency matrix A_r . The entire graph can be summarized by an adjacency tensor $A \in \mathbb{R}^{|V| \times |R| \times |V|}$, where R is the set of relations. One example of a multi-relational graph is the drug-drug interaction graph in which each node denotes a drug, and each edge's type corresponds to a side effect that can occur when a patient takes the pair of drugs simultaneously. Multi-relational graphs can be further classified as *heterogeneous* and *multiplex* graphs, where the set of nodes can be partitioned into disjoint sets $V = V_1 \cup V_2 \cup \dots \cup V_k$ where $V_i \cap V_j = \emptyset, \forall i \neq j$:

- In heterogeneous graphs, a node v in set V_i is labeled with the discrete node type i , and edges generally satisfy constraints according to the node types such that certain edges only connect nodes of certain types. One example of a heterogeneous graph is a biomedical graph in which one node type represents drugs, the another represents diseases and the last one represents proteins; edges representing “polypharmacy side-effects” would only occur between two drug nodes, the “treatment” edges connect between diseases nodes and drug nodes, while drug-protein/protein-protein interaction

and disease-protein association would be labeled by different edge types, respectively. One another example of a heterogeneous graph is the *multi-layer perceptron* (MLP) where there are multiple disjoint layers of neurons or nodes, and an edge representing a connection or synapse between a neuron with another neuron in the next layer.

- In multiplex graphs, the structure can be decomposed into a set of k layers: every node belongs to every layer; while each layer corresponds to a unique relation, representing the *intra-layer* edge type for that layer; and *inter-layer* edges connect the same node across layers. For example, in a multiplex transportation network, each node represents a city, each layer represents a different mode of transportation (e.g., via car, train or airplane, etc.), intra-layer edges represent pairs of cities connected by a particular mode of transportation, and inter-layer edges represent the switching modes of transporation within a city (e.g., from metro to bus, etc.).

We can also have *attribute* or *feature* information associated with a graph. The real-valued node-level features can be represented by a matrix $F_{\text{node}} \in \mathbb{R}^{|V| \times d}$, where each node is associated with a d -dimensional vector of information. For example, in molecular graphs, node features might include the atom type (e.g., Carbon, Hydrogen or Oxygen, etc.) and other chemical features of atoms. Furthermore, the real-valued edge-level features can be represented by a tensor $F_{\text{edge}} \in \mathbb{R}^{|V| \times |V| \times d'}$. For instance, one of the edge features in a molecular graph is the bond type. For convenience, F_{node} matrix can be stored in the diagonal of F_{edge} tensor, if we view a node feature as a self-loop edge feature. Again, if the edge features are discrete (such as the bond type in molecular graphs), one can represent the graph similarly as for multi-relational graphs.

Lastly, a *hypergraph* is a generalization of a graph in which an edge can join any number of vertices. Formally, an undirected hypergraph H is a pair $H = (V, E)$ where V is the set of nodes, and E is a set of non-empty subsets of V , called *hyperedges*. Thus, E is a subset

of $\mathcal{P}(V) \setminus \{\emptyset\}$, where $\mathcal{P}(V)$ denotes the *power set* of V . One example of a hypergraph is the drug combination network in which each node is a drug, and a hyperedge connecting n nodes (v_1, v_2, \dots, v_n) might have hyperedge labels “side-effect” or “treatment” as results when a patient taking this combination of n drugs at the same time. Another example of a hypergraph is knowledge graphs which store facts using relations between multiple entities. A comprehensive summary of graph formalism can be found in (Lovász, 2012) and (Hamilton, 2020).

1.1.2 Learning tasks on graphs

Within artificial intelligence and machine learning, there are three basic approaches: *supervised learning*, *self-supervised learning*, and *unsupervised learning*. Supervised learning is a machine learning approach that is defined by its use of labeled datasets that are designed to “supervise” or train the learning algorithms or models to classify data or predict outcomes accurately. In particular, supervised learning can be separated into two types of problems: *regression* that uses an algorithm (e.g., linear regression, support vector regression, etc.) to understand the relationship between independent and dependent variables; and *classification* that uses an algorithm (e.g., linear classifiers, multi-layer perceptron, support vector machines, decision trees, random forest, etc.) to assign data samples into specific categories or classes. On the other hand, unsupervised learning uses learning or optimization algorithms to analyze, cluster unlabeled data, and to discover hidden patterns in data without human intervention (e.g., the labeling job done manually by humans). Two main tasks of unsupervised learning are: *dimensionality reduction* that uses an algorithm (e.g., principal component analysis, singular value decomposition, etc.) to reduce the number of features or dimensions of data inputs to a manageable size while preserving the data integrity; and *clustering* that uses an algorithm (e.g., K-Means, spectral clustering, etc.) to group unlabeled data based on their differences and similarities. Finally, self-supervised learning (SSL) is an

intermediate form between supervised and unsupervised learning, and is based on artificial neural networks learning in two steps: first, the network weights are initialized by learning on some pseudo-labels; and second, the actual learning task is performed with supervised or unsupervised learning. In self-supervised learning, the training data can be divided into positive and negative examples. One of the most wellknown methods in this category is *Constrastive SSL* that uses both positive and negative examples in which the learning's loss function minimizes the distance between positive samples while maximizing the distance between negative ones. For example, the algorithm of contrastive SSL minimizes the distance between a sample (e.g., an image) and its transformed version by operations such as rotation, translation, scaling, etc.; while maximizing the distance between two samples with two different labels.

Machine learning on graphs, in particular, can be also divided into the usual categories of supervised, unsupervised and self-supervised learning. However, machine learning problems on graphs often blur the boundaries between the traditional machine learning categories. This section provides a brief overview of the most important machine learning tasks on graph data including:

- **Graph classification & regression:** Given a dataset of multiple different graphs, our goal is to make independent predictions specific to each graph. For example, given a dataset of molecular graphs, we might want to build a regression model that is able to accurately predict each molecule's toxicity, solubility, or some properties produced by the computationally expensive Density Functional Theory calculation. This problem can be seen as a supervised one.
- **Node classification:** Suppose we are given a citation network with millions of scientific papers represented as nodes and the directed edges represent if a paper cites another one. Each paper is associated with a feature vector capturing the frequency of

most important words. A small percentage of papers are classified into basic categories (e.g., Science, Technology, Education, Mathematics, etc.). The node classification task here is to predict the labels of the rest of not-yet-classified papers. Another example is learning on a large social network with millions of users in which there are acquaintance relationships among users, and we want to cluster users into groups or a hierarchy of groups. This problem is a mix of both supervised and unsupervised learning, and is sometimes referred as *semi-supervised learning*.

- **Link prediction:** The standard setup is that we are given a set of nodes and an incomplete set of edges between these nodes, and our goal is to use this partial information to infer the missing edges. For example, we might want to predict the missing citations in a partial citation network, or the missing friendship relations in an incomplete social network, or the unknown side-effects or potential new purposes of existing approved drugs, etc. Again, these problems can be understood as semi-supervised learning problems.

1.1.3 Machine learning methods on graphs

Kernel-based algorithms, such as Gaussian processes (Mackay, 1997), support vector machines (Burges, 1998), and kernel PCA (Mika et al., 1998), have been widely used in the statistical learning community. Graph kernel methods are approaches to designing features for graphs or implicit kernel functions that can be further used in machine learning models such as support vector machines and Gaussian processes. Learning on graphs has a long history in the kernels literature, including approaches based on random walks (Gärtner, 2002; Borgwardt and Kriegel, 2005; Feragen et al., 2013), counting subgraphs (Shervashidze et al., 2009), spectral ideas (Vishwanathan et al., 2010), label propagation schemes with hashing (Shervashidze et al., 2011; Neumann et al., 2016), and even algebraic ideas (Kondor and Borgwardt, 2008). Many of these papers address moderate size problems in chemo-

and bioinformatics, and the way they represent graphs is essentially fixed. To apply kernel methods on graphs, the proposed kernels must be *positive semi-definite*, which is usually achieved by summing up all pairs of local neighborhoods between two graphs, that has at least quadratic time- and quadratic space-complexity with respect to the number of nodes. Furthermore, a limitation of kernel methods, in general, is compute the kernel gram matrix by an algorithm of quadratic time- and quadratic space-complexity with respect to the number of examples in a dataset. All in all, the approach of graph kernels is computationally expensive and definitely infeasible to large-scale graph datasets.

One of the most successful graph kernels is the Weisfeiler–Lehman graph kernel, inspired from the Weisfeiler–Lehman graph isomorphism test in graph theory, which aims to build a multi-level, hierarchical representation by iteratively hashing the sub-structures of a graph (Shervashidze et al., 2011). Graph neural networks (GNNs) are indeed an extension from the construction of Weisfeiler–Lehman algorithm such that the *fixed* hashing function is replaced by a *learnable* one as a non-linearity mapping. The networks are designed to be totally differentiable and can be optimized by back-propagation and stochastic gradient descents in a data-driven manner. Graph neural networks can be seen as a way of generalizing the convolution concept to graphs. The most wellknown form of GNNs is message passing neural networks (MPNNs) Gilmer et al. (2017) that are built based on the message passing scheme in which each node propagates and aggregates information, encoded by vectorized messages, to and from its local neighborhood.

To represent graphs properly, the proposed model must be *permutation-invariant*, meaning that the outcome of the model stays the same (i.e. invariant) regardless of how we permute the order of nodes in the input graph. To preserve the permutation invariance, MPNNs and its variants propose a specific message passing’s aggregation where each node sums the mes-

sages coming from its neighborhood. We argue that this aggregation causes the fundamental limitation of this line of work. This aggregation restricts the representation power of MPNNs such as each node loses their identity after being aggregated (by summing). Therefore, we propose a new general architecture called *Covariant Compositional Networks* (CCNs) (Kondor et al., 2018a) (Hy et al., 2018) in which the messages are represented by higher order tensors and transform covariantly/equivariantly according to a specific representation of the symmetry group of its receptive field, in our case, the group of permutation \mathbb{S}_n . We generalize the aggregation to tensor contraction and tensor product operations such that the whole network preserves the permutation equivariance.

Density functional theory (DFT) is the workhorse of modern quantum chemistry due to its ability to calculate many properties of molecular systems with high accuracy. However, DFT is too computationally expensive for applications such as chemical search and drug screening, which may involve hundreds of thousands of compounds. By representing each molecule as a graph, CCNs successfully learns to predict molecular properties produced by DFT calculation in QM9 (Ramakrishnan et al., 2014) and CEP (Hachmann et al., 2011) datasets with high accuracies while being computationally efficient. To efficiently implement graph neural networks including CCNs, I implemented GraphFlow (Hy and Jones, 2019) (Hy, 2017), a Deep Learning framework built from scratch in C++/CUDA that supports automatic differentiation, dynamic computation graphs, and efficient tensor/matrix operations accelerated by GPU.

Chapter 3 includes our proposals of CCNs and GraphFlow, along with a detailed summary of the field of graph learning including various forms of graph kernels and graph neural networks, and several experiments on graph learning.

1.2 Deep generative models on graphs

The goal of *graph generation* is to build models that can generate realistic graph structures. Indeed, we will see that this problem is closely related to graph representation learning. Instead of being given a graph structure $G = (V, E)$ as the input, in graph generation we want the output of our model to be a graph G that has certain desirable properties.

1.2.1 Traditional approaches

First, we begin with a discussion of traditional approaches to graph generation, that are rooted from graph theory and mathematics, and actually pre-date modern machine learning research in general and graph representation learning. The traditional approaches to graph generation define how the edges in a graph are constructed, or in other words, specify a fixed *generative process*. We frame this generative process as a way of specifying the probability or likelihood $P(A_{v_1, v_2} = 1)$ of an edge between two nodes v_1 and v_2 .

Erdős–Rényi Model

Erdős–Rényi (ER) model is the most well-known and simplest random graph generative model that defines the likelihood of an edge between any pair of nodes as

$$P(A_{v_1, v_2} = 1) = p, \quad \forall v_1, v_2 \in V, \quad v_1 \neq v_2, \tag{1.1}$$

where $p \in [0, 1]$ is the parameter that controls the edge density of the generated graph (Erdos and Renyi, 1960). In this case, we assume that the edge probabilities are all independent. To generate a random graph, the ER model executes the following steps:

1. Choose (or sample) the number of nodes $N = |V|$,
2. Set the density parameter $0 < p < 1$,

3. Use Eq. 1.1 to sample every entry of the adjacency matrix.

The time complexity to generate a graph is $O(N^2)$, i.e., linear in the size of the adjacency matrix. Despite its simplicity, the shortcoming of the ER model is its failure to generate complex graphs that have structure and function of graphs in reality.

Stochastic Block Models

Stochastic Block Models (SBMs) seek to improve the ER model by extending it to generating graphs with community structure. The set of nodes V can be partitioned into B different blocks V_1, \dots, V_B . Every node $v \in V$ has a probability p_i of assigning to block i , i.e.

$$p_i = P(v \in V_i), \quad \forall v \in V, \quad i = 1, \dots, B;$$

and

$$\sum_{i=1}^B p_i = 1.$$

Block-to-block edge probabilities are specified by a symmetric matrix $C \in [0, 1]^{B \times B}$, where $C_{i,j}$ gives the probability of an edge occurring between a node in block V_i and a node in block V_j . The generative process is detailed as follows:

- For every node $v \in V$, we assign v to a block V_i by sampling from the categorical distribution (p_1, \dots, p_B) .
- For every pair of nodes $v_1 \in V_i$ and $v_2 \in V_j$, we sample an edge between v_1 and v_2 according to

$$P(A_{v_1, v_2} = 1) = C_{i,j}.$$

If $i = j$, basically we sample an “internal” edge of a block.

In SBMs, we can control the edge probabilities within and between different blocks, and this extension of the ER model allows us to generate graphs that exhibit community structure.

Preferential Attachment

Preferential Attachment (PA) proposed by (Albert and Barabási, 2002) is based on the assumption that real-world graphs exhibit *power law* degree distribution, in particular, the probability of a node v having degree d_v is given by

$$P(d_v = k) \propto k^{-\alpha}, \quad \alpha > 1.$$

Intuitively, the *heavy tailed* power distribution leads to a large number of nodes with small degrees, but a small number of nodes with extremely large degrees. The generative process is described as follows:

1. Suppose that we are going to generate a graph of n nodes. First, we initialize a fully connected graph with $m_0 < n$ nodes.
2. Iteratively add $n - m_0$ nodes to this graph. At iteration t , we add a new node u to the graph, and we connect u to $m < m_0$ existing nodes according to the following distribution:

$$P(A_{u,v} = 1) = \frac{d_v^{(t)}}{\sum_{v' \in V^{(t)}} d_{v'}^{(t)}},$$

where $d_v^{(t)}$ denotes the degree of node v at iteration t and $V^{(t)}$ denotes the set of nodes that have been added to the graph up to iteration t .

This generation process is called *autoregressive*, meaning that the graph is constructed gradually or incrementally by adding new nodes or new edges iteratively. In some sense, the PA's generation process is similar to the Dirichlet Process in which high-degree nodes tend to have more connections.

1.2.2 Data-driven approaches

There are two fundamental limitations of the traditional methods in graph generation:

1. They rely on a fixed, hand-crafted generation process. Thus, they do not possesss the ability to generate real-world graphs.
2. They are not a learning algorithm and lack the capability to learn a generative model from data.

Therefore, in this introduction, we present *data-driven approaches* based on Deep Learning that seek to learn a generative model of graphs based on a set of training graphs $\{G_1, \dots, G_n\}$. In general, these approaches have been referred to as *deep generative models on graphs*, that adopt the most popular generative models in machine learning such as variational autoencoders (VAEs) (Kingma and Welling, 2014), generative adversarial networks (GANs) (Goodfellow et al., 2014) and autoregressive models, for the ultimate goal of generating graphs with similar characteristics as the training set. There are two styles of graph generation:

1. **All-at-once:** This style generates the whole graph or adjacency matrix *at once*. Generally, VAEs and GANs follow this style such that the whole adjacency matrix is the output of the decoder or generator, respectively. These all-at-once generative models are analogous to the ER and SBM models in that we sample all edges in the graph simultaneously.
2. **Autoregressive:** This style generate a graph *incrementally* instead of all-at-once (e.g., generating graph node-by-node or edge-by-edge), that bear similarities to the PA model in that the probability of adding an edge at each step depends on what edges were previously added to the graph. The methods adopting Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) for graph generation have the autoregressive nature.

It is important to note that the autoregressive models do *not* preserve the permutation-invariance. Given a graph of N nodes, there are $N!$ different ways to construct this graph in the autoregressive manner. During training, models in the autoregressive category must define a particular ordering of nodes for the decoding process of reconstructing the input graph or adjacency matrix. In contrast, models in the all-at-once category, especially VAEs, can be designed to be permutation-equivariant such that if one permutes the order of nodes in the input graph, or even permute the node latents, the reconstructed graph/adjacency would be permuted/transformed accordingly in a completely predictable way. It is also possible to mix VAEs & GANs with an autoregressive decoder/generator that iteratively constructs a graph in multiple steps given a latent or a random vector, respectively. Even though, the encoder in VAEs and the discriminator in GANs must be permutation-invariant with respect to node ordering. We will discuss both types of graph generation in detailed as follows.

Graph Variational Autoencoders

Motivated from the theory of variational inference (Wainwright and Jordan, 2005), graph variational autoencoders as an extension of the conventional VAEs generally have three main components:

1. *Probabilistic encoder* $q_\theta(Z|G)$ defines a distribution over *latent representations*. We specify the latent conditional distribution as $Z \sim \mathcal{N}(\mu_\theta(G), \sigma_\theta(G))$, where $\mu_\theta(\cdot)$ and $\sigma_\theta(\cdot)$ are graph neural networks that generate the mean and variance parameters for a normal distribution that we will sample latent embeddings Z from. It is important to note that the Gaussian distribution here is isotropic, meaning that $\sigma_\theta(G)$ is indeed the diagonal of a digonal covariance matrix Σ .
2. *Probabilistic decoder* $p_\theta(A|Z)$, from which we can sample realistic graphs (i.e. adjacency matrices) $\hat{A} \sim p_\theta(A|Z)$ by conditioning on a latent variable Z .
3. *Prior distribution* $p(Z)$ over the latent space. We assume that $Z \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. Dur-

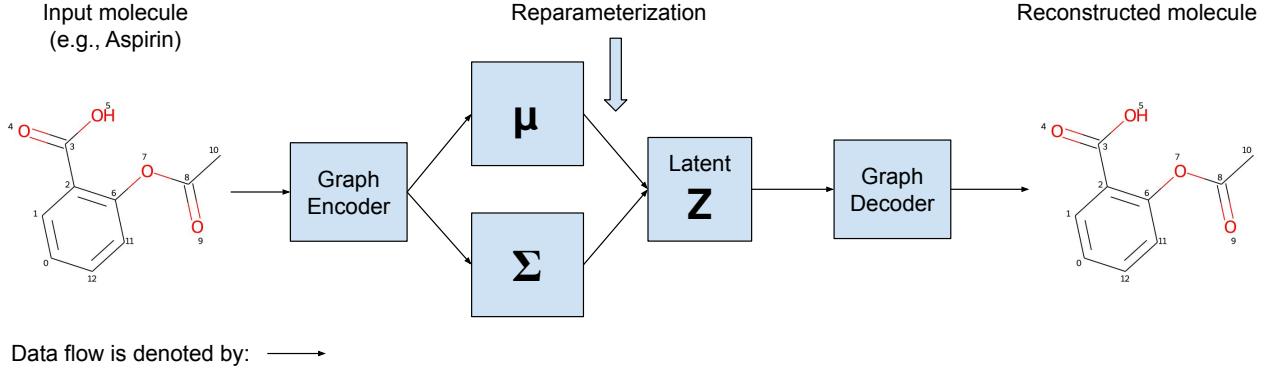


Figure 1.3: Graph variational autoencoder for molecule generation.

ing the training phase, the *reparameterization trick* to sample the latent Z without breaking the differentiability is simply as

$$Z = \mu_\theta(G) + \sigma_\theta(G) \odot \epsilon,$$

where \odot denotes the element-wise multiplication, and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ can be sampled easily. During the testing or generation phase, we basically sample Z from the prior and run the probabilistic decoder to get a generated graph.

Figure 1.3 depicts the architecture of a simple graph VAEs. Given a set of training graphs $\{G_1, \dots, G_n\}$, we can train a VAE model by minimizing the evidence likelihood lower bound (ELBO):

$$\mathcal{L} = \sum_{G_i \in \{G_1, \dots, G_n\}} \mathbb{E}_{q_\theta(Z|G_i)} [p_\theta(G_i|Z)] - \text{KL}(q_\theta(Z|G_i) \| p(Z)).$$

The basic idea of ELBO optimization is to maximize the reconstruction ability of our decoder $\mathbb{E}_{q_\theta(Z|G_i)} [p_\theta(G_i|Z)]$, while minimizing the Kullback–Leibner (KL) divergence between our posterior latent distribution $q_\theta(Z|G_i)$ and the prior $p(Z)$.

There are two types of latent representations: *node-level latents* and *graph-level latents*, each would have its own architectures. For the first one, the graph encoder generates latent rep-

resentations for each node in the graph; while for the second one, the graph encoder only produces a single vectorized latent for the whole graph. The idea of node-level latents was firstly introduced by (Kipf and Welling, 2016) and termed the Variational Graph Autoencoder (VGAE). However, the authors proposed VGAE model *not* as a generative model to sample new graphs. In this case, the encoder has two separate GNNs to generate mean and variance parameters, conditioned on the input adjacency A and node features F :

$$\mu_Z = \text{GNN}_\mu(A, F) \in \mathbb{R}^{|V| \times d},$$

$$\log \sigma_Z = \text{GNN}_\sigma(A, F) \in \mathbb{R}^{|V| \times d}.$$

The reparameterization trick to sample a set of latent node embeddings $Z \in \mathbb{R}^{|V| \times d}$ is $Z = \epsilon \odot \exp(\log(\sigma_Z)) + \mu_Z$, $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. The graph decoder employs a simple dot-product decoder:

$$p_\theta(A_{u,v} = 1 | z_u, z_v) = \gamma(z_u^T z_v),$$

where

$$\gamma(x) = \frac{1}{1 + e^{-x}}$$

is the element-wise sigmoid function. The graph likelihood is simply as

$$p_\theta(G|Z) = \prod_{(u,v) \in V \times V} p_\theta(A_{u,v} = 1 | z_u, z_v).$$

On the another hand, the idea of graph-level latents was originally proposed by (Simonovsky and Komodakis, 2018) and termed as GraphVAE. In contrast of VGAE defining a posterior distribution for each node, GraphVAE only has a single graph-level embedding $z_G \sim \mathcal{N}(\mu_{z_G}, \sigma_{z_G})$. The encoder of GraphVAE is defined as:

$$\mu_{z_G} = \text{POOL}_\mu(\text{GNN}_\mu(A, F)) \in \mathbb{R}^{|V| \times d},$$

$$\sigma_{z_G} = \text{POOL}_\sigma(\text{GNN}_\sigma(A, F)) \in \mathbb{R}^{|V| \times d},$$

where $\text{POOL} : \mathbb{R}^{|V| \times d} \rightarrow \mathbb{R}^d$ denotes a pooling function that maps a matrix of node embeddings $Z \in \mathbb{R}^{|V| \times d}$ to a graph-level embedding vector $z_G \in \mathbb{R}^d$. With the Bernoulli distributional assumption of edge probabilities, the decoder uses a multi-layer perceptron (MLP) to map the latent vector z_G to a matrix $\hat{A} \in [0, 1]^{|V| \times |V|}$ of edge probabilities: $\hat{A} = \gamma(\text{MLP}(z_G))$, that obviously does not respect the permutation equivariance. The posterior distribution in this case is

$$p_\theta(G|z_G) = \prod_{(u,v) \in V \times V} \hat{A}_{u,v} A_{u,v} + (1 - \hat{A}_{u,v})(1 - A_{u,v}),$$

where A denotes the true adjacency, and \hat{A} denotes the predicted edge probabilities. Key challenges of the graph-level latents approach are:

1. Because of the assumption of a fixed number of nodes, one must specify empirical distribution of graph sizes from the training data.
2. Because we do not know the correct ordering of nodes, to define the permutation-invariant posterior distribution, one must employ the **NP-hard** graph matching from A to \hat{A} (i.e. maximizing the graph likelihood over the set of all possible permutation Π of nodes):

$$p_\theta(G|z_G) = \max_{\pi \in \Pi} \prod_{(u,v) \in V \times V} \hat{A}_{u,v}^\pi A_{u,v} + (1 - \hat{A}_{u,v}^\pi)(1 - A_{u,v}), \quad (1.2)$$

where \hat{A}^π denotes the permuted adjacency matrix by permutation π (i.e. $\hat{A}^\pi = P_\pi \hat{A} P_\pi^T$ where P_π denotes the permutation matrix representing π). Since Eq. 1.2 is computationally infeasible to compute, one approximate solution is to specify a particular ordering function π :

$$p_\theta(G|z_G) \approx \prod_{(u,v) \in V \times V} \hat{A}_{u,v}^\pi A_{u,v} + (1 - \hat{A}_{u,v}^\pi)(1 - A_{u,v}),$$

or a bit smarter, we specify some small n permutations:

$$p_\theta(G|z_G) \approx \frac{1}{n} \sum_{\pi_i \in \{\pi_1, \dots, \pi_n\}} \prod_{(u,v) \in V \times V} \hat{A}_{u,v}^{\pi_i} A_{u,v} + (1 - \hat{A}_{u,v}^{\pi_i})(1 - A_{u,v}).$$

Generative Adversarial Networks for Graphs

The basic idea of GAN-based generative models is to

- Define a trainable *generator network* $g_\theta : \mathbb{R}^d \rightarrow \mathcal{X}$ that takes a random seed $z \in \mathbb{R}^d$ as input, and generates realistic (but fake) data samples $\hat{x} \in \mathcal{X}$. Here, \mathcal{X} denotes the space of all possible examples, either real or fake.
- Define a *discriminator network* $d_\phi : \mathcal{X} \rightarrow [0, 1]$ that outputs the probability a given input is fake, or distinguishing between real data samples $x \in \mathcal{X}$ and samples generated by the generator \hat{x} .

Both the generator and discriminator optimized jointly in an *adversarial game* or a mini-max optimization:

$$\min_{\theta} \max_{\phi} \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log(1 - d_\phi(x))] + \mathbb{E}_{z \sim p_{\text{seed}}(z)} [\log(d_\phi(g_\theta(z)))]$$

where $p_{\text{data}}(x)$ denotes the empirical distribution of real data samples, and $p_{\text{seed}}(z)$ denotes the random seed distribution (e.g. standard multivariate Gaussian). For graph generation, (Cao and Kipf, 2018) proposed MolGAN in which the generator generates a matrix of edge probabilities given a seed vector z : $\hat{A} = \gamma(\text{MLP}(z))$. Given this matrix of edge probabilities, we generate a discrete adjacency matrix $A \in \mathbb{Z}^{|V| \times |V|}$ by sampling independent Bernoulli variables for each edge $A_{u,v} \sim \text{Bernoulli}(\hat{A}_{u,v})$. The discriminator of MolGAN is any GNN-based graph classification model. The limitation of GAN-based models is training mini-max optimization is difficult and unstable.

Autoregressive Models

To model edge dependencies, we define the likelihood of a graph given a latent representation z by decomposing the overall likelihood into a set of independent edge likelihoods:

$$p(G|z) = \prod_{(u,v) \in V \times V} p(A_{u,v}|z).$$

In autoregressive approach, we assume that edges are generated sequentially and that the likelihood of each edge can be conditioned on the edges that have been previously generated:

$$p(G|z) = \prod_{i=1}^{|V|} p(L_{v_i}|L_{v_1}, \dots, L_{v_{i-1}}, z),$$

where L_{v_i} denotes the lower-triangle portion of the adjacency matrix A in the construction step when we add the i -th node. Here, we introduce some of the most common autoregressive models. For example, (You et al., 2018a) proposed GraphRNN employing recurrent neural networks (RNNs):

- Graph-level RNN maintains a hidden state h_i , which is updated after generating each row of the adjacency matrix L_{v_i} :

$$h_{i+1} = \text{RNN}_{\text{graph}}(h_i, L_{v_i})$$

- Node-level RNN generates the entries of L_{v_i} in an autoregressive manner. In particular, RNN_{node} takes the graph-level hidden state h_i as an initial input, and then sequentially generates the binary values of L_{v_i} assuming a conditional Bernoulli distribution for each entry.

Another instance is GRAN (i.e. graph recurrent attention networks), suggested by (Liao et al., 2019), modeling the conditional distribution of each row of the adjacency matrix by running a GNN on the graph that has been generated so far.

Multiresolution Equivariant Graph Variational Autoencoders

Learning to generate graphs with deep generative models is a difficult problem because graphs are combinatorial objects that typically have high order correlations between their discrete substructures (subgraphs) (You et al., 2018a) (Li et al., 2018) (Liao et al., 2019) (Liu et al., 2019) (Dai et al., 2020). Graph-based molecular generation (Gómez-Bombarelli et al., 2018) (Simonovsky and Komodakis, 2018) (Cao and Kipf, 2018) (Jin et al., 2018) (Thiede et al., 2020) involves further challenges, including correctly recognizing chemical substructures, and importantly, ensuring that the generated molecular graphs are chemically valid.

We can see that the theoretical bottleneck of non-equivariant generative models for graphs is the exponentially-large number of ways to construct the same graph given a latent representation. Obviously, non-equivariant models do not exploit the permutation symmetry. The another limitation of all existing graph generative models is the lack of higher-order latent representation that is more expressive and carries more information from the encoder to the decoder. However, higher-order latents require a higher-order prior that models the correlation between nodes. It is trivial the conventional prior used in literature, standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{1})$ assuming the independence between node latents, lacks this ability to capture node correlation. Furthermore, we want to a generative model that is able to generate graphs in multiple resolutions with the graph encoder capable of capturing the hierarchical multiscale structures of graphs, that is especially important for molecules since a molecular graph might contain several functional groups as subgraphs. Therefore, one of the objectives of this thesis is to design a multiresolution and end-to-end equivariant graph generative model built upon hierarchical variational autoencoder with the following criteria:

1. End-to-end permutation equivariant generative model with equivariant encoder, decoder, latent representation and also prior function;

2. Higher-order latent representation, and the corresponding higher-order prior function;
3. Multiscale graph encoder that can capture hierarchical structures and builds hierarchical latent structures.

In this thesis, we propose *Multiresolution Graph Networks* (MGN) and *Multiresolution Graph Variational Autoencoders* (MGVAE) (Hy and Kondor, 2021b) (Thiede et al., 2020) to learn and generate graphs in a multiresolution and equivariant manner. At each resolution level, MGN employs higher order message passing to encode the graph while learning to partition it into mutually exclusive clusters and coarsening into a lower resolution. MGVAE constructs a hierarchical generative model based on MGN to variationally autoencode the hierarchy of coarsened graphs. Our proposed framework is end-to-end permutation equivariant with respect to node ordering. Our methods have been successful with several generative tasks including link prediction on citation graphs, unsupervised molecular representation learning to predict molecular properties, molecular generation, general graph generation and graph-based image generation. Details of our proposals are in Chapter 4.

1.2.3 Evaluating graph generation

It is a challenging task to evaluate the quality of graph generation. The current practice is to analyze different statistics of the generated graphs, and to compare the distribution of statistics for the generated graphs to a test set. Assume we have a set of graph statistics $S = (s_1, s_2, \dots, s_n)$ where each of these statistics $s_{i,G} : \mathbb{R} \rightarrow [0, 1]$ is assumed to define a univariate distribution over \mathbb{R} for a given graph G . For example, the graph statistics can be:

- Degree distribution,
- Distribution of clustering coefficients,
- Distribution of different motifs or graphlets.

Given a particular statistic s_i , total variance distance between the statistics of a test graph $s_{i,G_{\text{test}}}$ and a generated graph $s_{i,G_{\text{gen}}}$:

$$d(s_{i,G_{\text{test}}}, s_{i,G_{\text{gen}}}) = \sup_{x \in \mathbb{R}} |s_{i,G_{\text{test}}}(x) - s_{i,G_{\text{gen}}}(x)|$$

To measure the performance, we compute the average pairwise distributional distance between a set of generated graphs and graphs in a test set (e.g. Wasserstein distance).

Molecule generation is a special case of graph generation that further imposes more challenges:

- Graph structures need to be both valid (e.g. chemically stable), and ideally have some desirable properties (e.g. medicinal properties, or solubility).
- Unlike general graph generation, molecule generation can benefit substantially from domain-specific knowledge for both model design and evaluation strategies.

Chapter 4 includes our proposals of MGN, MGVAE and its learning to cluster algorithm, along with many experiments on molecular graph generation, general graph generation, link prediction on citation graphs, semi-supervised molecular representation and graph-based image generation.

1.3 Group equivariant molecular neural networks

1.3.1 Symmetry-preserving neural networks

Symmetry in mathematics is a type of invariance: the property that a mathematical object remains unchanged under a set of operations or transformations. Such transformations can be either smooth, continuous, or discrete. Symmetries are important in many scientific problems and machine learning tasks, for instance:

- In graph representation learning, the scalar output of any graph neural networks $\phi_\theta(G)$, where θ denotes the set of all learnable parameters, must be invariant with respect to the permutation symmetry of the input graph, or in other words $\phi_\theta(G) = \phi_\theta(G')$ if G is isomorphic to G' (i.e. $G \simeq G'$). Furthermore, any graph kernels $\kappa(G_1, G_2)$ between two graphs must respect permutation-invariance (i.e. $\kappa(G_1, G_2) = \kappa(G'_1, G'_2)$ for every $G_1 \simeq G'_1$ and $G_2 \simeq G'_2$).
- In quantum chemistry, any neural networks predicting the molecular properties must be rotationally invariant with respect to the molecule's orientation in space. Suppose we are given a three-dimensional conformational geometry $R \in \mathbb{R}^{n \times 3}$ of a molecule of n atoms, and a *molecular neural network*¹ $\psi_\theta(\cdot)$. Rotational invariance implies that $\psi_\theta(R) = \psi_\theta(Rr)$ for every 3×3 rotation matrix r . Indeed, ψ_θ must respect both rotation and permutation symmetries, meaning that in addition, we must have $\psi_\theta(R) = \psi_\theta(PR)$ for every $n \times n$ permutation matrix P . In physics, groups are especially important because they describe the symmetries which the laws of physics seem to obey.
- In computer graphics and computer vision, the convolutional neural networks predicting the category of an object given its image respect the translational invariance. Obviously, the object category is unchanged by shifts.

In group theory, the collection of all symmetries form an algebraic object known as a *group* or *symmetry group*. The set of symmetries of an object satisfies a number of properties:

- If both transformations leave the object invariant, then so does the composition of transformations, and thus the composition is also a symmetry. In other words, if x and y are two symmetries, then their compositions $x \circ y$ and $y \circ x$ are also symmetries². The

1. We use this term to refer to any neural network architecture taking the input as a molecule and make prediction for its molecular properties.

2. We will follow the notation convention in group theory, $x \circ y = xy$ that reads from right to left as: we first apply y and then x .

order of transformations in a composition is important, as in many cases symmetries are non-commutative (i.e. $xy \neq yx$). In group theory, commutative groups (i.e. $xy = yx$ for every pair of symmetries (x, y)) are called Abelian ³.

- Symmetries are always *invertible*, and the inverse is also a symmetry. The inverse of a symmetry x is denoted as x^{-1} . It is trivial to see that simultaneously applying a symmetry and its inverse leaves the object unchanged and $xx^{-1} = x^{-1}x$.

Formal definitions and detailed discussion regarding group theory are included in the appendix chapter 9. The groups of interest for graph neural networks and graph kernels is the permutation group, denoted as \mathbb{S}_n ; for molecular neural networks is both three-dimensional rotation group $\text{SO}(3)$ and \mathbb{S}_n ; and for convolutional neural networks is the group of two-dimensional integer translations, \mathbb{Z}^2 . These types of neural networks are called *symmetry-preserving* or respecting the underlying symmetries of its input data and domain.

In order to build symmetry-preserving models, we are mostly interested in how groups *act* on data, rather than considering groups as abstract entities. We assume that there is a domain Ω underlying our data and we consider a group G . A *group action* of G on a set Ω is defined as a mapping $(g, u) \mapsto g \cdot u$ associating a group element $g \in G$ and a point $u \in \Omega$ with some other point on Ω in a way that is compatible with the group operations, i.e.:

- $u = g \circ (g^{-1} \cdot u) = g^{-1} \circ (g \cdot u)$ for all $g \in G$ and $u \in \Omega$;
- $g \circ (h \cdot u) = (g \circ h) \cdot u$ for all $g, h \in G$ and $u \in \Omega$;
- If G is Abelian then $g \circ (h \cdot u) = h \circ (g \cdot u)$ for all $g, h \in G$ and $u \in \Omega$.

The most important kind of group actions is *linear* group actions, also known as *group representation*. We will encounter group & representation interchangeably ⁴ throughout this thesis. A linear action g on signals on the domain Ω is understood in the sense that

3. Named after the Norwegian mathematician Niels H. Abel.

4. As physicists usually do!

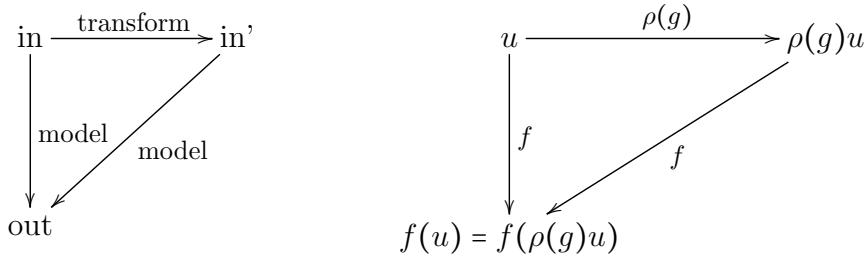


Figure 1.4: Demonstration of the definition of a G -invariant function in the right. The output is unchanged with respect to transformation of the input (see the simplified diagram in the left).

$g \cdot (\alpha u + \beta u') = \alpha(g \cdot u) + \beta(g \cdot u')$ for any scalars $\alpha, \beta \in \mathbb{R}$ and signals $u, u' \in \Omega$. We can describe linear actions either as maps $(g, u) \mapsto g \cdot u$ that are linear in u , or equivalently, as a map $\rho : G \rightarrow \mathbb{R}^{n \times n}$ that assigns to each group element g an $n \times n$ *invertible* matrix $\rho(g)$, satisfying the condition $\rho(g \circ h) = \rho(g)\rho(h)$ for all $g, h \in G$ ⁵. The assignment of matrices to group elements is indeed equivalent to the linear group action, and the matrix representing a composite group element gh (i.e. $\rho(gh)$) is equal to the matrix product of the representation of g and h . In essence, a representation makes an abstract algebraic object more concrete by describing its elements by matrices and their algebraic operations, that leads to the birth of *Representation theory*, which is an important branch of mathematics that studies abstract algebraic structures by representing their elements as linear transformations of vector spaces, and studies modules over these abstract algebraic structures. The appendix chapter 9 introduces briefly the most important concepts and results in the vast subject of representation theory.

Symmetry of the signals' domain Ω imposes structure on the function f defined on such signals. This thesis studies two important cases: *invariant* and *equivariant* functions. A function $f : \Omega \rightarrow \mathcal{X}$ is G -*invariant* if $f(\rho(g)u) = f(u)$ for all $g \in G$ and $u \in \Omega$; i.e. regardless

5. The dimension n of the matrix is in general arbitrary and not necessarily related to the dimensionality of the group G . In applications, n is usually the dimensionality of the feature space on which the group acts on.

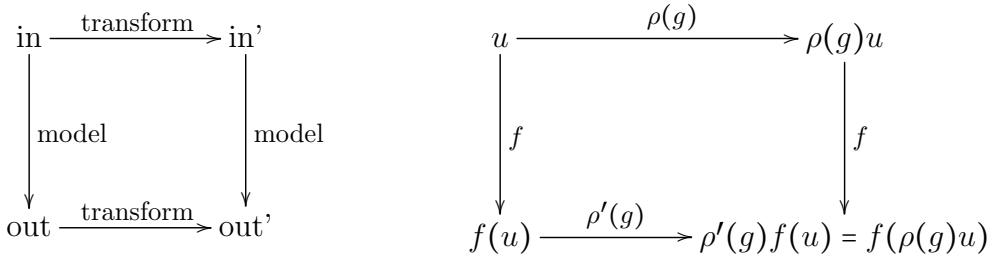


Figure 1.5: Demonstration of the definition of a G -equivariant or G -covariant function. The output is transformed according to the transformation of the input (see the simplified diagram in the left).

of how we transform the input by a linear group action (or multiplying with a group representation), the function's outcome in some space \mathcal{X} is unchanged (see Fig. 1.4). On the other hand, a function $f : \Omega \rightarrow \Omega'$ is *G -equivariant* or *G -covariant* if $f(\rho(g)u) = \rho'(g)f(u)$ for all $g \in G$, where ρ and ρ' are representations of the same group G (indeed, $\rho(\cdot)$ and $\rho'(\cdot)$ are linear operators in Ω and Ω' , respectively). In other words, G -equivariant means the group action on the input affects the output in a completely predictable way (see Fig. 1.5). Indeed, G -invariant is just a special case of G -equivariant.

A neural network ϕ , in general, can be written as a composition of multiple functions $\{f_1, \dots, f_L\}$ in which function f_ℓ is the ℓ -th layer out of L layers of the network⁶ as follows:

$$\phi = f_L \circ f_{L-1} \circ \dots \circ f_1,$$

where f_1 is the bottom layer directly taking the input data and f_L is the top or output layer making the prediction. An G -equivariant or G -covariant neural network is the one composing of all G -equivariant layers. For classification or regression tasks, since the model outputs scalar values, the top layer f_L is G -invariant; while for graph generation, we need all components to be G -equivariant. For example, our *Covariant Compositional Networks*

6. For simplicity, we assume there is no skip connection, and the network is just layer-by-layer.

(see Chapter 3) predicting molecular properties and our *Multiresolution Equivariant Graph Variational Autoencoders* (see Chapter 4) are permutation-equivariant (i.e. $G = \mathbb{S}_n$ specifically). For simplicity in writing, sometimes we can drop “ G ” but the symmetry G can be determined based on the context of the problem/task.

1.3.2 Rotational equivariant neural networks

The power of convolutional neural networks (CNNs) is rooted from their ability to exploit the translational symmetries through translation equivariance, as we briefly mentioned. It is natural to consider generalizations that exploit other groups of symmetries such as our works detailed in chapters 3 & 4 for permutation equivariance mainly in the context of graph learning and graph generation. For physical and chemical applications, rotation equivariance is equally important as permutation equivariance in designing neural network architectures that are able to learn the behavior and properties of complex many-body systems or molecular systems in which each atom or particle is located in 3D space. For example, it is necessary to capture the 3D geometry or the tertiary structure of a protein to make a robust prediction for the protein function. Furthermore, CNNs are restricted to 2D planar images, while various applications in computer vision, robotics and global climate modelling demand models that can analyze spherical images which can be understood as functions on the unit sphere S^2 or quotient $\text{SO}(3)/\text{SO}(2)$. In any case, we must handle signals on rotation group $\text{SO}(3)$ and find our way to generalize the convolution for this group. This introduction presents the most wellknown methods in the area including *Spherical CNNs* (Cohen et al., 2018) and *Clebsch-Gordan Nets* (Kondor et al., 2018b) that pave the way for our general architecture, *Covariant Molecular Neural Networks*, applied in the molecular setting (Anderson et al., 2019) that will be discussed in details in chapter 5. First, we will define the correlations (i.e. convolutions) between functions on S^2 and $\text{SO}(3)$, the generalized fast Fourier transform to efficiently compute the correlations with the help from the convolutional theorem, and the

architecture of Spherical CNNs. After that, we will introduce the Clebsch–Gordan transform on group $\text{SO}(3)$ as a form of nonlinearity and define the architecture of Clebsch–Gordan Nets, the generalized Spherical CNNs learning completely on the Fourier domain.

Correlations on S^2 and $\text{SO}(3)$

The theoretical backbone of both (Cohen et al., 2018) and (Kondor et al., 2018b) in achieving rotation equivariance is how to define and efficiently perform correlations or cross-correlations (i.e. convolution in deep learning language) on the unit sphere and rotation group by using generalized Fourier Transform (FT) and its generalized Fast Fourier Transform (FFT) algorithms. This section introduces the necessary mathematical background, while a deeper insight of the generalized FT & FFT is described in appendix chapter 9.

We define S^2 or the unit sphere, which technically is not a group, as the set of points $x \in \mathbb{R}^3$ with norm 1 (i.e. $\|x\|_2^2 = 1$), that is a two-dimensional manifold which can be parameterized by spherical coordinates $\alpha \in [0, 2\pi)$ and $\beta \in [0, \pi)$. We model spherical images and filters as continuous spherical signals/functions $f : S^2 \rightarrow \mathbb{R}^C$, where C is the number of channels. Remark that each channel $f_c : S^2 \rightarrow \mathbb{R}$ is indeed a spherical function itself, and all of our definitions below will work similarly for the single-channel case (i.e. $C = 1$). We define group $\text{SO}(3)$ or the special orthogonal group as the set of rotations in three dimensions in which each rotation can be represented by a 3×3 orthogonal matrix that preserves orientation (i.e. $\det(R) = 1$) and distance (i.e. $\|Rx\| = \|x\|$). Furthermore, $\text{SO}(3)$ is a three-dimensional manifold which can be specified by the common convention of ZYZ Euler angles given by $\alpha \in [0, 2\pi)$, $\beta \in [0, \pi)$ and $\gamma \in [0, 2\pi)$, which quantify the rotation around the Z-, X- and Z-axes performed in the order, respectively. We introduce the rotation operator T_R (i.e. group action) that takes a spherical function f and produces a rotated one $T_R f$ as $[T_R f](x) \triangleq f(R^{-1}x)$ for all $x \in S^2$. Trivially, we have $T_{RR'} = T_R T_{R'}$. The inner

product between two spherical functions is defined as

$$\langle h, f \rangle_{S^2} \triangleq \int_{S^2} \sum_{c=1}^C h_c(x) f_c(x) dx = \sum_{c=1}^C \langle h_c, f_c \rangle_{S^2},$$

where dx denotes the rotation invariant integration measure on the unit sphere, which can be expressed as $d\alpha \sin(\beta) d\beta / 4\pi$ in spherical coordinates. For simplicity in writing, we remove the subscription for the inner product notation. For any rotation $R \in \text{SO}(3)$, the invariance of the measure ensures that $\int_{S^2} f(Rx) dx = \int_{S^2} f(x) dx$. In addition, we have

$$\langle T_R h, f \rangle = \int_{S^2} \sum_{c=1}^C h_c(R^{-1}x) f_c(x) dx = \int_{S^2} \sum_{c=1}^C h_c(x) f_c(Rx) dx = \langle h, T_{R^{-1}} f \rangle,$$

that implies $T_{R^{-1}}$ is adjoint to T_R and T_R is unitary. For spherical functions h and f , we define the spherical correlation as

$$[h * f](R) \triangleq \langle T_R h, f \rangle = \int_{S^2} \sum_{c=1}^C h_c(R^{-1}x) f_c(x) dx. \quad (1.3)$$

It is important to note that the output of the spherical correlation defined in Eq. 1.3 is a function on $\text{SO}(3)$. Thus, our next step is to define the rotation group correlation. Suppose we are given a function on the rotation group $f : \text{SO}(3) \rightarrow \mathbb{R}^C$ and a rotation $R \in \text{SO}(3)$, the rotation operator T_R acting on f is defined by $[T_R f](Q) \triangleq f(R^{-1}Q)$ for all $Q \in \text{SO}(3)$. Using the same analogy as spherical correlation in Eq. 1.3, the correlation of two functions on the rotation group $h, f : \text{SO}(3) \rightarrow \mathbb{R}^C$ is defined as

$$[h * f](R) \triangleq \langle T_R h, f \rangle_{\text{SO}(3)} = \int_{\text{SO}(3)} \sum_{c=1}^C h_c(R^{-1}Q) f_c(Q) dQ,$$

where $\langle \cdot, \cdot \rangle_{\text{SO}(3)}$ denotes the inner product on the rotation group in which dQ is the invariant measure, which can be expressed in ZYZ-Euler angles as $d\alpha \sin(\beta) d\beta d\gamma / (8\pi^2)$. We can show

that the rotation group correlation preserves the rotational equivariance as follows:

$$[h * T_Q f](R) = \langle T_R h, T_Q f \rangle = \langle T_{Q^{-1}R} h, f \rangle = [h * f](Q^{-1}R) = T_Q[h * f](R),$$

that means if we transform the input function f then the correlation $h * f$ will be transformed accordingly. The spherical correlation can be also proven to be equivariant in the same manner.

Generalized Fast Fourier Transform

The Fourier theorem states that $\widehat{h * f} = \hat{h} \cdot \hat{f}$. For a complex signal in \mathbb{C}^n , Fast Fourier Transform (FFT) has the time complexity of $O(n \log n)$ instead of $O(n^2)$ algorithm of Discrete Fourier Transform (DFT). Because the product \cdot operation can be done by a linear $O(n)$ algorithm, the computation of correlation can be done efficiently by using FFT. For functions on the unit sphere and rotation group, we will need the generalized Fourier transform on groups (GFT) and its corresponding fast algorithm (GFFT). Mathematically, GFT is a linear projection onto a set of orthogonal basis functions called *matrix element of irreducible unitary representations*. For the real numbers \mathbb{R} , these are the complex exponentials. For S^2 , these are the spherical harmonics $Y_m^\ell(x)$ that is indexed by $\ell \geq 0$ and $-\ell \leq m \leq \ell$, and has $2\ell + 1$ elements. For the rotation groups $\text{SO}(3)$, the irreducible representations are the *Wigner D-matrices* as a function of the (α, β, γ) Euler angles:

$$D_{m,n}^\ell(\alpha, \beta, \gamma) = \sqrt{\frac{4\pi}{2\ell+1}} (Y_m^\ell(\beta, \alpha))^* e^{-in\gamma},$$

where $Y_m^\ell(\beta, \alpha)$ denotes the spherical harmonics; and $D_{m,n}^\ell$ is indexed by $\ell \geq 0$ and $-\ell \leq m, n \leq \ell$, and has dimension $(2\ell+1) \times (2\ell+1)$. Interestingly, the spherical harmonics Y_m^ℓ can be obtained by taking the middle column of the Wigner D-matrix $D_{m,0}^\ell$. We can write down the GFT of a single-channel function defined on these manifolds as follows; while for the

multiple channels, we can compute the GFT for each channel simultaneously. The Fourier coefficients of a bandlimited spherical signal f with a bandwidth of L are given by

$$\hat{f}_m^\ell = \int_{S^2} f(x) \bar{Y}_m^\ell(x) dx, \quad (1.4)$$

and the corresponding inverse Fourier transform or the synthesis function is given by

$$f(x) = \sum_{\ell=0}^L \sum_{m=-\ell}^{\ell} \hat{f}_m^\ell Y_m^\ell(x).$$

On the other hand, the Fourier coefficients of a signal f defined on $\text{SO}(3)$ with a bandwidth of L is given by

$$\hat{f}_{m,n}^\ell = \int_{\text{SO}(3)} f(x) \bar{D}_{m,n}^\ell(x) dx, \quad (1.5)$$

and the inverse $\text{SO}(3)$ Fourier transform is given by

$$f(R) = \sum_{\ell=0}^L (2\ell+1) \sum_{m=-\ell}^{\ell} \sum_{n=-\ell}^{\ell} \hat{f}_{m,n}^\ell D_{m,n}^\ell(R).$$

The GFT in Eq. 1.4 and Eq. 1.5 can be computed efficiently by the GFFT algorithm proposed by (Kostelec and Rockmore, 2007), and the correlation between two signals h and f can be computed using the correlation theorem as

$$\widehat{[h * f]}_\ell = \hat{f}_\ell \cdot \hat{h}_\ell^\dagger, \quad \ell \in \{0, 1, \dots, L\},$$

where \dagger denotes the conjugate transpose (i.e. Hermitian conjugate) operation, and \hat{f}_ℓ and \hat{h}_ℓ are vectors and matrices for S^2 and $\text{SO}(3)$, respectively.

Spherical CNNs

The general architecture of spherical convolutional neural networks (spherical CNNs) pro-

posed by (Cohen et al., 2018) can be described as follows. Given an input spherical signal $f^{(0)} : S^2 \rightarrow \mathbb{R}^{C_0}$, where C_0 is the number of channels in the input (e.g., $C_0 = 3$ channels indicating red/green/blue of colorized spherical image). The input spherical signal was sampled on a discrete grid. The first convolutional layer performs a correlation on S^2 , between the input spherical signal $f^{(0)}$ and C_1 filters $\{h_i^{(1)} : S^2 \rightarrow \mathbb{R}^{C_0}\}_{i=1}^{C_1}$; and then apply a nonlinearity (e.g., ReLU, sigmoid, etc.) to produce the output of the first layer, a signal $f^{(1)} : \text{SO}(3) \rightarrow \mathbb{R}^{C_1}$ defined on the rotation group. The rest of the convolutional layers in the model are $\text{SO}(3)$ correlations where the filters of the d -th layer ($d > 1$) are $\{h_i^{(d)} : \text{SO}(3) \rightarrow \mathbb{R}^{C_{d-1}}\}_{i=1}^{C_d}$. After D convolutional layers, we obtain the feature map $f^{(D)} : \text{SO}(3) \rightarrow \mathbb{R}^{C_D}$. The input of the $\text{SO}(3)$ FFT, for all convolutional layers, is a spatial signal sampled on a discrete grid and stored as a 3D array. Finally, we use an integration layer to construct the rotation-invariant feature $z \in \mathbb{R}^{C_D}$ as

$$z_i = \int_{\text{SO}(3)} f_i^{(D)}(x) dx, \quad i \in \{1, \dots, D\}.$$

On top of the network, we apply a multilayer perceptron or a fully connected layer to make the regression/classification prediction.

Clebsch–Gordan transform & generalized spherical CNNs

(Kondor et al., 2018b) proposes the generalization of spherical CNNs as an architecture learning completely on the Fourier domain and introduces the use of covariant nonlinearities by the Clebsch–Gordan transform. These techniques will be useful for us to construct our covariant molecular neural networks later on. If we rotate a spherical function f by some $R \in \text{SO}(3)$ as $[T_R f](x) = f(R^{-1}x)$, then each vector of its S^2 Fourier transform just gets multiplied with the corresponding Wigner D-matrix:

$$\hat{f}_\ell \mapsto \rho_\ell(R) \cdot \hat{f}_\ell,$$

where $\rho_\ell(R) \in \mathbb{C}^{(2\ell+1) \times (2\ell+1)}$ denotes the Wigner D-matrix⁷. For functions on $\text{SO}(3)$, the situation is the same. If $f : \text{SO}(3) \rightarrow \mathbb{C}$ and f' is the rotated function $f'(Q) = [T_R f](Q) = f(R^{-1}Q)$, then the Fourier matrices of f' are $\hat{f}'_\ell = \rho_\ell(R)\hat{f}_\ell$. Proposition 1.3.1 shows that outputs of the cross-correlations behave analogously. Definition 1.3.1 defines the *generalized $\text{SO}(3)$ -covariant spherical CNN* in which each neuron or Fourier matrix transforms covariantly/equivariantly with respect to rotations. It is important to note that the original spherical CNN is indeed a special case of this generalization in which $\tau^{(0)} = (1, 1, \dots, 1)$ and $\tau^{(d)} = (1, 3, \dots, 2\ell + 1, \dots, 2L + 1)$ for $d \geq 1$.

Proposition 1.3.1. *Let $f : S^2 \rightarrow \mathbb{C}$ be an spherical function transformed under the action T_R of a rotation R , and $h : S^2 \rightarrow \mathbb{C}$ be a filter. Then, each Fourier component of the cross-correlation transforms as*

$$\widehat{[h * f]}_\ell \mapsto \rho_\ell(R) \cdot \widehat{[h * f]}_\ell.$$

*Similarly, if $f', h' : \text{SO}(3) \rightarrow \mathbb{C}$, then their cross-correlation $\widehat{h' * f'}$ transforms the same way.*

Definition 1.3.1. Let \mathcal{N} be an D layer feed-forward neural network whose input is a C_0 -channel spherical function $f^{(0)} : S^2 \rightarrow \mathbb{C}^{C_0}$. We say that \mathcal{N} is a *generalized $\text{SO}(3)$ -covariant spherical CNN* if the output of each layer d can be expressed as a collection of vectors

$$\hat{f}^{(d)} = \left(\hat{f}_0^{(d)}, \hat{f}_1^{(d)}, \dots, \hat{f}_L^{(d)} \right), \quad (1.6)$$

7. We use this ρ notation to be consistent with the one used for irreducible representations in Representation Theory.

in which each part $\hat{f}_\ell^{(d)}$ is also a collection of vectors

$$\hat{f}_\ell^{(d)} = \left(\hat{f}_{\ell,1}^{(d)}, \hat{f}_{\ell,2}^{(d)}, \dots, \hat{f}_{\ell,\tau_\ell^{(d)}}^{(d)} \right),$$

where each $\hat{f}_{\ell,j}^{(d)} \in \mathbb{C}^{2\ell+1}$ is a ρ_ℓ -covariant vector in the sense that if the input signal is rotated by some rotation R , then $\hat{f}_{\ell,j}^{(d)}$ transforms as

$$\hat{f}_{\ell,j}^{(d)} \mapsto \rho_\ell(R) \cdot \hat{f}_{\ell,j}^{(d)}.$$

We call the individual $\hat{f}_{\ell,j}^{(d)}$ vectors the irreducible *fragments* of $\hat{f}^{(d)}$, and the integer vector $\tau^{(d)} = (\tau_0^{(d)}, \tau_1^{(d)}, \dots, \tau_L^{(d)})$ counting the number of fragments for each ℓ the *type* of $\hat{f}^{(d)}$.

In a covariant neural network architecture, the linear operation of each layer must be covariant. Proposition 1.3.2 shows us that if we stack all fragments of \hat{f} corresponding to the same ℓ into a $(2\ell+1) \times \tau_\ell^{(d)}$ dimensional matrix $F_\ell^{(d)}$, and we do the same of \hat{g} , then we have

$$G_\ell^{(d)} = F_\ell^{(d)} W_\ell^{(d)}, \quad \ell \in \{0, 1, \dots, L\},$$

for some sequence of complex value matrices $W_0^{(d)}, \dots, W_L^{(d)}$. Note that $W_\ell^{(d)}$ does not necessarily need to be square, i.e., the number of fragments in \hat{f} and \hat{g} corresponding to ℓ might be different. The entries of the $W_\ell^{(d)}$ matrices are learnable parameters that we can compute the gradients for via back-propagation, if and only if the whole network (i.e. every component in it) is differentiable, and then optimize by stochastic gradient descent.

Proposition 1.3.2. *Let $\hat{f}^{(d)}$ be an $SO(3)$ -covariant activation function of the form 1.6 in Def. 1.3.1, and $\hat{g}^{(d)} = \mathcal{L}(\hat{f}^{(d)})$ be a linear function of $\hat{f}^{(d)}$ written in a similar form. Then $\hat{g}^{(d)}$ is $SO(3)$ -covariant if and only if each $\hat{g}_{\ell,j}^{(d)}$ fragment is a linear combination of fragments from $\hat{f}^{(d)}$ with the same ℓ .*

In representation theory, the Clebsch–Gordan decomposition arises in the context of decomposing the tensor product or Kronecker product of irreducible representations into a direct sum of irreducibles. For the rotation group $\text{SO}(3)$, it takes from

$$\rho_{\ell_1}(R) \otimes \rho_{\ell_2}(R) = C_{\ell_1, \ell_2} \left[\bigoplus_{\ell=|\ell_1-\ell_2|}^{\ell_1+\ell_2} \rho_\ell(R) \right] C_{\ell_1, \ell_2}^T, \quad R \in \text{SO}(3),$$

where C_{ℓ_1, ℓ_2} are fixed matrices, and \oplus denotes the Dirac sum or concatenating matrices into a block-diagonal one. Equivalently, letting $C_{\ell_1, \ell_2, \ell}$ denote the appropriate block of columns of C_{ℓ_1, ℓ_2} ,

$$\rho_\ell(R) = C_{\ell_1, \ell_2, \ell}^T [\rho_{\ell_1}(R) \otimes \rho_{\ell_2}(R)] C_{\ell_1, \ell_2, \ell}.$$

Tensor $C_{\ell_1, \ell_2, \ell}$ is actually sparse, in particular $[C_{\ell_1, \ell_2, \ell}]_{m_1, m_2, m} = 0$ unless $m_1 + m_2 = m$. We will need to precompute the Clebsch–Gordan coefficients $C_{\ell_1, \ell_2, \ell}$ before any computation.

Proposition 1.3.3. *Let \hat{f}_{ℓ_1} and \hat{f}_{ℓ_2} be two ρ_{ℓ_1} and ρ_{ℓ_2} covariant vectors, respectively; and ℓ be any integer between $|\ell_1 - \ell_2|$ and $\ell_1 + \ell_2$. Then*

$$\hat{g}_\ell = C_{\ell_1, \ell_2, \ell}^T [\hat{f}_{\ell_1} \otimes \hat{f}_{\ell_2}] \quad (1.7)$$

is a ρ_ℓ -covariant vector.

Based on Proposition 1.3.3, the generalized spherical networks computes the Clebsch–Gordan decomposition of the tensor product between two Fourier matrices/fragments as in Eq. 1.7 as the nonlinearity. In matrix notation, we can write it as

$$G_\ell^{(d)} = \bigsqcup_{|\ell_1 - \ell_2| \leq \ell \leq \ell_1 + \ell_2} C_{\ell_1, \ell_2, \ell}^T [F_{\ell_1}^{(d)} \otimes F_{\ell_2}^{(d)}],$$

where \sqcup denotes merging matrices horizontally. This operation increases the size of the activation substantially: the total number of fragments is squared. To avoid the exponential

space, following the Clebsch–Gordan transform as the nonlinearity, we apply another learnable linear transformation that reduces the number of fragments for each ℓ to some fixed maximum number $\bar{\tau}_\ell$.

After the D -th layer, the activations of the network will be a sequence of matrices $F_0^{(D)}, \dots, F_L^{(D)}$, each transforming covariantly under rotations according to $F_\ell^{(D)} \mapsto \rho_\ell(R)F_\ell^{(D)}$. But to make a prediction for a regression/classification task, we need the output to be invariant with respect to rotations instead, i.e. a collection of scalars. Therefore, for the final layer, we only take the elements of $F_0^{(D)}$ that are invariant and apply a multilayer perceptron or a fully connected layer for the downstream task. This step completes our construction for the generalized $\text{SO}(3)$ -covariant spherical CNN. However, this so-called “generalized” network only applies for a single spherical signal as the input. In setting of the molecular learning, there are multiple atoms and each atom is associated with its own spheical signal. In some sense, the spherical network can be understood as a model learning on a single particle. It is definitely a challenge to design a molecular neural network that is covariant with respect to both permutation of atoms and rotation of its coordinates. Fortunately, we can reuse the foundation of the Clebsch–Gordan transform as the nonlinearity from the spherical network in our new molecular neural networks.

Covariant Molecular Neural Networks

In the line of group equivariant models learning molecules, our *Covariant Molecular Neural Networks* (Cormorant) model (Anderson et al., 2019) is the first neural network architecture in which the operations implemented by the neurons is directly motivated by the form of known physical interactions. Rotation and translation invariance are explicitly “baked into” the network by the fact all activations are represented in spherical tensor form ($\text{SO}(3)$ -vectors), and the neurons combine Clebsch–Gordan products, concatenation of parts and

mixing with learnable weights, all of which are covariant operations. In future work, we envisage the potentials learned by Cormorant to be directly integrated in molecular dynamics simulation frameworks. In this regard, it is very encouraging that on MD-17, which is the standard benchmark for force field learning, Cormorant outperforms all other competing methods. Learning from derivatives (forces) and generalizing to other compact symmetry groups are natural extensions of this work.

Chapter 5 includes our proposal of Cormorant, and experiments on learning atomic potential energy surfaces for use in molecular dynamics simulations, and learning ground state properties of molecules calculated by Density Functional Theory. Furthermore, chapter 9 is an appendix chapter that includes a brief introduction to group and representation theory.

1.4 Multiresolution Machine Learning

From the begining of this introduction, we have seen that graphs appear at every scale in Nature, from gigantic structures such as galaxies with diameter measured in light years to the quantum world of atoms with diamter measured in nanometers⁸. Humans' understanding of the Universe, as far as the advancement of Science, seems to be structured in multiple resolutions such that for each resolution, we need a different theory. For instance, for massive objects such as galaxies, stars, black holes or things moving at nearly the speed of light, we need the Albert Einstein's general theory of relativity; while for our daily stuffs, the classical Newtonian physics is accurate enough. But at the scale of atoms and subatomic particles like electrons, protons and neutrons, none of these theories worked, so mathematicians and physicists developed the theory of quantum mechanics that is the foundation of all quantum physics including quantum chemistry, quantum field theory, quantum computation

8. It is a matter of fact that we still don't know if the Universe is infinite and what the smallest particle is.

and quantum information. However, the two theories forming our current understanding of physics, general theory of relativity and quantum mechanics, seem not to work together conceptually: in the former one, events are deterministic; while in the latter one, events produced by the particles happen with probability rather than definite outcomes. This is indeed an example to show us that choosing the right resolution or the right frame to work on is not a trivial task. The ultimate goal of modern physics might possibly be a unified *theory of everything* that fully explains and links together all aspects of the Universe, and of course works fine on every resolution.

With the above motivation, this thesis defines the term *Multiresolution Machine Learning* for the first time. Multiresolution Machine Learning refers to the group or class of machine learning algorithms operating on multiple levels of resolutions of the inputs, modeling the relations between resolutions and between objects in the same resolution to make a robust and consistent prediction. To illustrate the distinction between multiresolution and the traditional sense of machine learning, our proposals in graph learning and graph generation in chapter 4, *Multiresolution Graph Network* (MGN) and *Multiresolution Equivariant Graph Variational Autoencoders* (MGVAE) can serve as an example. In the supervised setting, MGN constructs a hierarchy of graph resolutions by iteratively learning to partition a graph into multiple clusters and coarsening it into the higher level of resolution. On each level, MGN employs a separate graph neural network to encode the corresponding resolution into the corresponding latent representation. Consequentially, MGN builds a hierarchy of latents along with a hierarchy of coarsened graphs. For the downstream task, a combination of multiresolution latents is informative to make a thorough prediction, that is especially important for highly structured objects such as molecular graphs in which functional groups like the Benzene rings usually form a cluster in the higher level of resolution, and complex networks with community structures in which each densely connected community tends to

group together into a single cluster. Experimentally, MGN outperforms the current state-of-the-art graph neural networks in estimating the solubility of drug-like molecules; while at the same time, is able to detect the functional groups responsible for the prediction in a complete data-driven manner. In the computational aspect, for large and hierarchical molecular structures such as proteins, it is necessary to employ a multiresolution method as MGN to coarsen the primary structure into a compressed representation which still preserves the overall geometric shape of the protein and reduces the computational cost of the whole neural network. Furthermore, MGN in combination with hierarchical variational autoencoders allows us to generate graphs in multiple resolutions, that results into the construction of MGVAE. In graph generation, MGVAE is the first model that is multiresolution in nature and also end-to-end permutation equivariant. In the experiments, MGVAE outperforms or gets an equivalent result in comparison with other state-of-the-art methods in both general graph generation and molecular graph generation. MGVAE has shown that it can also learn the molecular representation in an unsupervised manner via autoencoding. A simple multilayer perceptron learning on the unsupervised molecular vectorized features outperforms the wellknown graph kernels. Furthermore, MGVAE brings a better result in missing link prediction on citation graphs which have hierarchical structures. Interestingly, MGVAE is also the first generative model generating images in multiresolution manner, if we represent an image as a two-dimensional grid graph and the coarsening operation is similar to rescaling the image.

In the field of image & signal processing, processing is more efficient and simpler in a sparse representation where fewer coefficients reveal the information that we are searching for. Based on this motivation, Multiresolution Analysis (MRA) has been proposed by (Mallat, 1989b) as a design for multiscale signal approximation in which the sparse representations can be constructed by decomposing signals over elementary waveforms chosen in a family called

wavelets. Besides Fourier transforms, the discovery of wavelet orthogonal bases has opened the door to new transforms such as continuous and discrete wavelet transforms and the fast wavelet transform algorithm that have become crucial for several computer applications. Inspired by MRA from image & signal processing, there are two fundamental and important questions this thesis wants to answer simultaneously:

1. A novel method for constructing sparse wavelet transforms of functions defined on the nodes of an arbitrary graph.
2. An efficient algorithm for multiresolution approximation of hierarchical matrices.

Interestingly, this thesis addresses both of them at the same time as follows, to add one more important part to the general picture of Multiresolution Machine Learning.

Multiresolution Matrix Factorization & Graph Wavelet Neural Networks

Large matrices appear to have complex hierarchical structures that traditional linear algebra methods based on the low rank assumption are unable to capture. *Multiresolution Matrix Factorization* (MMF) (Kondor et al., 2014) offers an alternative to the low rank paradigm by introducing multiresolution matrices that can capture structure at multiple different scales: $A \approx U_1^T U_2^T \dots U_L^T H U_L \dots U_2 U_1$ where $\{U_1, \dots, U_L\}$ are sparse orthogonal matrices and H is close to diagonal. However, the authors only proposed suboptimal heuristics to solve localized problems instead of directly tackling the global MMF optimization. To address this limitation, we propose a learning algorithm combining of Stiefel manifold optimization and Reinforcement Learning to directly and globally optimize MMF (Hy and Kondor, 2021a). Furthermore, based on the wavelet basis produced by MMF when factorizing the normalized graph Laplacian, we construct the *Wavelet Neural Networks* (WNNs) (Hy and Kondor, 2021a) learning on the spectral domain with the graph convolution defined by the sparse wavelet transform. We have shown that the wavelet basis resulted from our learnable MMF far outperforms prior MMF algorithms, and the corresponding wavelet networks yield state

of the art results on standard node classification on citation graphs and molecular graph classification.

Chapter 6 includes background of MMF, our proposal of the learning algorithm to solve MMF based on Stiefel manifold optimization and Reinforcement Learning, as well as our proposal of WNNs learning graphs based on sparse wavelet transform given the MMF wavelet basis, and experiments on matrix approximation, node classification on citation graphs, and molecular graph classification. Furthermore, chapter 10 is an appendix chapter that includes a brief introduction to multiresolution analysis, wavelet theory and spectral graph theory. Given these promising results, Multiresolution Machine Learning promises to improve the accuracies and computational efficiency of graph learning and generative models, and group equivariant neural networks for many other scientific problems as mentioned in chapter 7 of future works.

CHAPTER 2

THESIS OVERVIEW

2.1 General structure

This thesis is structured as follows:

- **Chapter 1 – Thesis Introduction** includes declaration of the thesis' objectives and a brief introduction for each topic.
- **Chapter 2 – Thesis Overview** includes the highlight of content for each chapter.
- **Chapter 3 – Graph Representation Learning** includes
 - Summary of the field of graph learning including various forms of graph kernels and graph neural networks,
 - Our proposal of *Covariant Compositional Networks* (CCNs),
 - Our proposal of GraphFlow deep learning framework in C++/CUDA and its performance evaluation,
 - And several experiments on molecular graph learning.
- **Chapter 4 – Multiresolution Equivariant Graph Variational Autoencoder** includes
 - Our proposal of *Multiresolution Graph Network* (MGN) and its learning to cluster algorithm,
 - Our proposal of *Multiresolution Equivariant Graph Variational Autoencoder* (MG-VAE),
 - And experiments on molecular graph generation, general graph generation, link prediction on citation graphs, semi-supervised molecular representation and graph-based image generation.
- **Chapter 5 – Rotationally Equivariant Molecular Neural Networks** includes
 - Background of group and representation theory,

- Our proposal of *Covariant Molecular Neural Networks* (Cormorant),
- And experiments on learning atomic potential energy surfaces for use in molecular dynamics simulations, and learning ground state properties of molecules calculated by Density Functional Theory.
- **Chapter 6 – Learning Multiresolution Matrix Factorization and its Wavelet Networks on Graphs** includes
 - Background of *Multiresolution Matrix Factorization* (MMF),
 - Our proposal of the learning algorithm to solve MMF based on Stiefel manifold optimization and Reinforcement Learning,
 - Our proposal of *Wavelet Neural Networks* (WNNs) learning graphs based on sparse wavelet transform given the MMF wavelet basis,
 - And experiments on matrix approximation, node classification on citation graphs, and molecular graph classification.
- **Chapter 7 – Future Works** includes
 - Potential applications of MGVAE in drug and material discovery,
 - Multiscale learning and generative models for large hierarchical structures such as proteins, and highly symmetric structures such as crystals,
 - Extension of MMF to tensor factorization and application in data visualization.
- **Chapter 8 – Thesis Conclusion** is the summary of discussions.
- **Chapter 9 – Appendix: Group & Representation Theory** is an appendix that includes
 - Introduction to group and representation theory,
 - Introduction to Fourier analysis, its efficient algorithms, the group theoretic perspective and quantum Fourier transform.
- **Chapter 10 – Appendix: Wavelet Theory** is an appendix that includes
 - Introduction to multiresolution analysis,

- Introduction to classical wavelet transform, spectral graph wavelet transform and diffusion wavelets.

2.2 Publications

This PhD thesis includes but not limited to the following publications of my prior works:

1. **Truong Son Hy** and Risi Kondor, *Learning Multiresolution Matrix Factorization and its Wavelet Networks on Graphs*, 2021.
Paper: <https://arxiv.org/pdf/2111.01940.pdf>
2. **Truong Son Hy** and Risi Kondor, *Multiresolution Equivariant Graph Variational Autoencoder*, 2021.
Paper: <https://arxiv.org/pdf/2106.00967.pdf>
3. Erik Henning Thiede, **Truong Son Hy** and Risi Kondor, *The general theory of permutation equivariant neural networks and higher order graph variational encoders*, 2020.
Paper: <https://arxiv.org/pdf/2004.03990.pdf>
4. Brandon Anderson, **Truong Son Hy** and Risi Kondor, *Cormorant: Covariant molecular neural networks*, Advances in Neural Information Processing Systems, 2019.
Paper: <https://arxiv.org/pdf/1906.04015.pdf>
5. **Truong Son Hy** and Chris Jones, *Graph neural networks with efficient tensor operations in CUDA/GPU and GraphFlow deep learning framework in C++ for quantum chemistry*, 2019.
Paper: <http://people.cs.uchicago.edu/~hytruongson/CCN-GraphFlow.pdf>
6. **Truong Son Hy**, Shubhendu Trivedi, Horace Pan, Brandon M. Anderson and Risi Kondor. *Predicting molecular properties with covariant compositional networks*, The

Journal of Chemical Physics 148.24, 2018.

Paper: <https://aip.scitation.org/doi/10.1063/1.5024797>

7. Risi Kondor, **Truong Son Hy**, Horace Pan, Brandon Anderson and Shubhendu Trivedi, *Covariant compositional networks for learning graphs*, International Conference on Learning Representations (workshop track), 2018.

Paper: <https://arxiv.org/pdf/1801.02144.pdf>.

8. **Truong Son Hy**, *Covariant compositional networks for learning graphs and Graph-Flow deep learning framework in C++/CUDA*, MSc Thesis at The University of Chicago, 2018.

Paper: <http://people.cs.uchicago.edu/~hytruongson/Master-Thesis.pdf>

Google Scholar:

<https://scholar.google.com/citations?user=xEXxSN4AAAAJ&hl=en>

2.3 Softwares

This PhD thesis includes experiments, tables, figures and results from the following softwares of my prior works:

1. **Learnable MMF**: Learning Multiresolution Matrix Factorization and its Wavelet Networks on Graphs.

https://github.com/risilab/Learnable_MMF

2. **MGVAE**: Multiresolution Equivariant Graph Variational Autoencoder (MGVAE) and Multiresolution Graph Networks (MGN) for supervised molecular properties prediction, unsupervised molecular representation learning, graph generation, citation link prediction and graph-based image generation.

<https://github.com/HyTruongSon/MGVAE>

3. LibCCNs: Covariant Compositional Networks Library is an easy-to-use and efficient implementation of Covariant Compositional Networks (CCNs) with TensorFlow and PyTorch's APIs based on a shared common C++ core.

<https://github.com/HyTruongSon/LibCCNs>

4. Invariant Graph Networks: A PyTorch implementation of the Invariant and Equivariant Graph Networks.

<https://github.com/HyTruongSon/InvariantGraphNetworks-PyTorch>

5. GraphFlow: Deep Learning framework built from scratch in C++/CUDA that supports symbolic/automatic differentiation, dynamic computation graphs, tensor/matrix operations accelerated by GPU and implementations of various state-of-the-art graph neural networks and other Machine Learning models.

<https://github.com/HyTruongSon/GraphFlow>

GitHub:

<https://github.com/HyTruongSon/>

<https://github.com/risilab>

CHAPTER 3

GRAPH REPRESENTATION LEARNING

3.1 Chapter Introduction

In the field of Machine Learning, standard objects such as vectors, matrices, tensors were carefully studied and successfully applied into various areas including Computer Vision, Natural Language Processing, Speech Recognition, etc. However, none of these standard objects are efficient in capturing the structures of molecules, social networks or the World Wide Web which are not fixed in size. This arises the need of graph representation and extensions of Support Vector Machine and Convolution Neural Network to graphs.

To represent graphs in general and molecules specifically, the proposed models must be *permutation-invariant* and *rotation-invariant*. In addition, to apply kernel methods on graphs, the proposed kernels must be *positive semi-definite*. Many graph kernels and graph similarity functions have been introduced by researchers. Among them, one of the most successful and efficient is the Weisfeiler-Lehman graph kernel which aims to build a multi-level, hierarchical representation of a graph (Shervashidze et al., 2011). However, a limitation of kernel methods (see section 3.2) is quadratic space usage and quadratic time-complexity in the number of examples. The common idea of family of Weisfeiler-Lehman graph kernel is hashing the sub-structures of a graph. Extending this idea, we come to the simplest form of graph neural networks in which the *fixed* hashing function is replaced by a *learnable* one as a non-linearity mapping (see section 3.4). We detail the graph neural network baselines such as Neural Graph Fingerprint (Duvenaud et al., 2015) and Learning Convolutional Neural Networks (Niepert et al., 2016) in sections 3.4.3 and 3.4.4. In the context of graphs, the sub-structures can be considered as a set of vertex feature vectors. We utilize the convolution operation by introducing higher-order representations for each vertex, from zeroth-order as

a vector to the first-order as a matrix and the second-order as a 3rd order tensor in section 3.7.2. Also in this chapter, we introduce the notions of tensor contractions and tensor products (see section 3.7.3) to keep the orders of tensors manageable without exponentially growing. Our generalized convolution graph neural network is named as Covariant Compositional Networks (CCNs) (Kondor et al., 2018a) (Hy et al., 2018).

Current Deep Learning frameworks including TensorFlow (Abadi et al., 2016), PyTorch (Paszke et al., 2017), Mxnet (Chen et al., 2016), Theano (Al-Rfou et al., 2016), etc. showed their limitations for constructing dynamic computation graphs along with specialized tensor operations. It leads to the need of a flexible programming framework for graph neural networks addressing both these drawbacks. With this motivation, we designed our Deep Learning framework in C++ named GraphFlow (Hy and Jones, 2019) for our long-term Machine Learning research. All of our experiments have been implemented efficiently within GraphFlow. In addition, GraphFlow is currently being parallelized with CPU/GPU multi-threading. Implementation of GraphFlow is mentioned in section 3.8. Finally, we apply our methods to the Harvard Clean Energy Project (HCEP) (Hachmann et al., 2011) and QM9 (Ramakrishnan et al., 2014) molecular dataset. The visualizations, experiments and empirical results are detailed in section 3.9.

3.2 Graph Kernels

Given the input domain \mathcal{X} that is some nonempty set, the common idea is to express the correlations or the similarities between pairs of points in \mathcal{X} in terms of a kernel function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ (Hofmann et al., 2008). The kernel function $k(\cdot, \cdot)$ is required to satisfy that for all $x, x' \in \mathcal{X}$,

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \quad (3.1)$$

where $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ maps from the input domain \mathcal{X} into some dot product space \mathcal{H} . We call Φ as a feature map and \mathcal{H} as a feature space. Given a kernel k and inputs $x_1, \dots, x_n \in \mathcal{X}$, the $n \times n$ matrix

$$K \triangleq (k(x_i, x_j))_{ij} \quad (3.2)$$

is called the Gram matrix (or kernel matrix) of kernel function k with respect to x_1, \dots, x_n . A symmetric matrix $K \in \mathbb{R}^{n \times n}$ satisfying

$$c^T K c = \sum_{i,j} c_i c_j K_{ij} \geq 0 \quad (3.3)$$

for all $c \in \mathbb{R}^n$ is called positive definite. If equality in 3.3 happens when $c_1 = \dots = c_n = 0$, then K is called strictly positive definite. A symmetric function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is called a positive definite kernel on \mathcal{X} if

$$\sum_{i,j} c_i c_j k(x_i, x_j) \geq 0 \quad (3.4)$$

holds for any $n \in \mathbb{N}$, $\{x_1, \dots, x_n\} \subseteq \mathcal{X}^n$ and $c \in \mathbb{R}^n$. The inequality 3.4 is equivalent with saying the Gram matrix K of kernel function k with respect to inputs $x_1, \dots, x_n \in \mathcal{X}$ is positive definite. Recall that in the continuous case, positive semi-definiteness amounts to

$$\int_{\mathcal{X}} \int_{\mathcal{X}} f(x) f(x') k(x, x') dx dx' \geq 0$$

for all square integrable real functions $f \in L_2(\mathcal{X})$, that is sometimes referred to as Mercer's condition.

A graph kernel $\mathcal{K}_{graph} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a positive definite kernel having the input domain \mathcal{X} as a set of graphs. Given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Assume that each vertex is associated with a feature vector $f : V \rightarrow \Omega$ where Ω is a vector space. A

positive definite graph kernel \mathcal{K}_{graph} between G_1 and G_2 can be written as:

$$\mathcal{K}_{graph} \triangleq \frac{1}{|V_1|} \cdot \frac{1}{|V_2|} \cdot \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} k_{base}(f(v_1), f(v_2)) \quad (3.5)$$

where k_{base} is any base kernel defined on vector space Ω , and can be:

- Linear: $k_{base}(x, y) \triangleq \langle x, y \rangle_{norm} \triangleq x^T y / (\|x\| \cdot \|y\|)$
- Quadratic: $k_{base}(x, y) \triangleq (\langle x, y \rangle_{norm} + q)^2$ where $q \in \mathbb{R}$
- Radial Basis Function (RBF): $k_{base}(x, y) \triangleq \exp(-\gamma \|x - y\|^2)$ where $\gamma \in \mathbb{R}$

We introduce some of the most well-known graph kernels.

3.2.1 Weisfeiler-Lehman graph kernel

Given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ where V_1 and V_2 are the sets of vertices, E_1 and E_2 are the sets of edges. Suppose we have a mapping $l : V_1 \cup V_2 \rightarrow \Sigma$ that is a vertex labeling function giving label $l(v) \in \Sigma$ from the set of all possible labels Σ for each vertex $v \in V_1 \cup V_2$. Assuming that $|V_1| = |V_2|$ and $|E_1| = |E_2|$, the graph isomorphism test is defined as: Determine whether there exists a permutation on the vertex indices such that two graphs G_1 and G_2 are identical. Formally saying, we have to find a bijection between the set of vertices of G_1 and G_2 , $\sigma : V_1 \rightarrow V_2$, such that

$$\forall (u, v) \in E_1 : (\sigma(u), \sigma(v)) \in E_2 \quad (3.6)$$

In addition, we can add one more constraint on the vertex labels such that

$$\forall v \in V_1 : l(v) = l(\sigma(v)) \quad (3.7)$$

The algorithm of Weisfeiler-Lehman (WL) graph isomorphism test (Weisfeiler and Lehman, 1968) is described as follows.

Algorithm 1 Weisfeiler-Lehman iterations

```

1: Input: Given an undirected graph  $G = (V, E)$ , vertex labels  $l(v) \in \Sigma$  for all  $v \in V$ , and  $T \in \mathbb{N}$  as the number of Weisfeiler-Lehman iterations. Assuming that we have a perfect hashing function  $h : \Sigma^* \rightarrow \Sigma$ .
2: Output: Return  $f_i(v) \in \Sigma^*$  for all  $v \in V$  and  $i \in [0, T]$ .
3: for  $v \in V$  do
4:    $M_i(v) \leftarrow \emptyset$ 
5:    $s_i(v) \leftarrow l(v)$ 
6:    $f_i(v) \leftarrow h(s_i(v))$ 
7: end for
8: for  $i = 1 \rightarrow T$  do
9:   Compute the multiset of labels  $M_i(v)$ , string  $s_i(v)$  and compressed label  $f_i(v)$ 
10:  for  $v \in V$  do
11:     $M_i(v) \leftarrow \{f_{i-1}(u) | u \in \mathcal{N}(v)\}$  where  $\mathcal{N}(v) = \{u | (u, v) \in E\}$ 
12:    Sort  $M_i(v)$  in ascending order and concatenate all labels of  $M_i(v)$  into string  $s_i(v)$ 
13:     $s_i(v) \leftarrow s_i(v) \oplus f_{i-1}(v)$  where  $\oplus$  is concatenation operation.
14:     $f_i(v) \leftarrow h(s_i(v))$ 
15:  end for
16: end for

```

We can see that if G_1 and G_2 are isomorphic then the WL test always returns true. In the case G_1 and G_2 are not isomorphic, the WL test returns true with a small probability. In particular, the WL algorithm has been shown to be a valid isomorphism test for almost all graphs (Babai and Kucera, 1979). Suppose that we have an efficient sorting algorithm $O(N \log_2 N)$ for a sequence of N items, and the time complexities for concatenation operations and computing the hashing functions are negligible. In algorithm 1, for each iteration i -th, each edge (u, v) is considered twice and $|\mathcal{N}(v)| \leq |V|$. Thus the time complexity of algorithm 1 is $O(T(|V| + |E| \log_2 |V|))$. In algorithm 2, $|F_i^{G_1}| = |V_1|$, the time complexity to sort $F_i^{G_1}$ is $O(|V_1| \log_2 |V_1|)$, and similarly for G_2 . Therfore, the total time comlexity of WL isomorphism test is $O(T(|V| + |E|) \log_2 |V|)$ where $|V| = \max\{|V_1|, |V_2|\}$ and $|E| = |E_1| + |E_2|$.

Algorithm 2 Weisfeiler-Lehman graph isomorphism test

- 1: **Input:** Given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with vertex labels $l : V_1 \cup V_2 \rightarrow \Sigma$.
- 2: **Output:** Return whether G_1 and G_2 are isomorphic.
- 3: Apply algorithm 1 on graph G_1 to get $\{f_i^{G_1}(v)\}$
- 4: Apply algorithm 1 on graph G_2 to get $\{f_i^{G_2}(v)\}$
- 5: **for** $i = 0 \rightarrow T$ **do**
- 6: $F_i^{G_1} \leftarrow \{f_i^{G_1}(v) | v \in V_1\}$
- 7: $F_i^{G_2} \leftarrow \{f_i^{G_2}(v) | v \in V_2\}$
- 8: Sort $F_i^{G_1}$ in ascending order
- 9: Sort $F_i^{G_2}$ in ascending order
- 10: **if** $F_i^{G_1} \neq F_i^{G_2}$ **then**
- 11: Return: G_1 and G_2 are **not** isomorphic.
- 12: **end if**
- 13: **end for**
- 14: Return: G_1 and G_2 are isomorphic.

Based on algorithms 1 2 and equation 3.5, we introduce the following algorithm to compute the Weisfeiler-Lehman kernel between the two input graphs G_1 and G_2 . In this case, G_1 and G_2 can have different numbers of vertices. The remaining question is: what would be the possible choices of vertex labels $l(v)$? One way to define the vertex labels is using the vertex degrees:

$$l(v) \triangleq |\{u | (u, v) \in E\}| = |\mathcal{N}(v)| \quad (3.8)$$

Suppose that the time complexity to compute the base kernel value between $f^{G_1}(v_1)$ and $f^{G_2}(v_2)$ is $O(T)$ for every pair of vertices (v_1, v_2) . Thus the time complexity to compute the WL kernel value is $O(T|V|^2)$ where $|V| = \max\{|V_1|, |V_2|\}$. Therefore, the total time complexity of WL graph kernel algorithm is $O(T(|V|^2 + |E| \log_2 |V|))$ where $|E| = |E_1| + |E_2|$.

3.2.2 Optimal assignment kernel and histogram-alignment WL graph feature

We define the *optimal assignment kernel* (Kriege et al., 2016) as follows. Let $[\mathcal{X}]^n$ denote the set of all n -element subsets of a set \mathcal{X} and $\mathcal{B}(X, Y)$ denote the set of all bijections between

Algorithm 3 Weisfeiler-Lehman graph kernel (Shervashidze et al., 2011)

- 1: **Input:** Given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with vertex labels $l : V_1 \cup V_2 \rightarrow \Sigma$.
 - 2: **Output:** Return the WL kernel value.
 - 3: Apply algorithm 1 on graph G_1 to get $\{f_i^{G_1}(v)\}$
 - 4: Apply algorithm 1 on graph G_2 to get $\{f_i^{G_2}(v)\}$
 - 5: **for** $v \in V_1$ **do**
 - 6: $f^{G_1}(v) \leftarrow \bigoplus_{i=0}^T f_i^{G_1}(v)$
 - 7: **end for**
 - 8: **for** $v \in V_2$ **do**
 - 9: $f^{G_2}(v) \leftarrow \bigoplus_{i=0}^T f_i^{G_2}(v)$
 - 10: **end for**
 - 11: $\mathcal{K}_{graph}(G_1, G_2) \leftarrow \frac{1}{|V_1|} \cdot \frac{1}{|V_2|} \cdot \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} k_{base}(f^{G_1}(v_1), f^{G_2}(v_2))$
 - 12: Return: $\mathcal{K}_{graph}(G_1, G_2)$
-

X, Y in $[\mathcal{X}]^n$ for $n \in \mathbb{N}$. The optimal assignment kernel $K_{\mathcal{B}}^k$ on $[\mathcal{X}]^n$ is defined as

$$K_{\mathcal{B}}^k(X, Y) \triangleq \max_{B \in \mathcal{B}(X, Y)} W(B)$$

where

$$W(B) \triangleq \sum_{(x,y) \in B} k(x, y)$$

and k is a base kernel on \mathcal{X} . In order to apply the kernel to sets of different cardinality, i.e. $|X| \neq |Y|$, we fill up the smaller set by additional objects z such that $k(x, z) = 0$ for all $x \in \mathcal{X}$. Finding the optimal B can be formalized as the Hungarian matching problem that can be solved efficiently by Kuhn-Munkres algorithm (Munkres, 1957). As following, we define the strong kernel and state its relation to the validity of an optimal assignment kernel.

Definition 3.2.1 (Strong kernel). A function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ is called **strong kernel** if $k(x, y) \geq \min\{k(x, z), k(z, y)\}$ for all $x, y, z \in \mathcal{X}$.

Theorem 3.2.1 (Validity of an optimal assignment kernel). *If the base kernel k is strong, then the function $K_{\mathcal{B}}^k$ is a valid kernel.*

Proof. Detail of the proof is contained in (Kriege et al., 2016). \square

Inspired by the histogram intersection kernel that yields an optimal assignment kernel (Kriege et al., 2016), we introduce the histogram alignment WL graph feature as follows. Suppose that the vertex labels can be discretized and encoded as one-hot vectors of size c . Let $\ell_v \in \{0, 1\}^c$ be the label vector of vertex v . For each vertex v , we consider the histogram of vertex labels of vertices at distance n from v as:

$$h_v^n = \sum_{w \in V : d(v, w) = n} \ell_w$$

where $d(v, w)$ denotes length of the shortest path between v and w . Given depth N , the histogram alignment WL graph feature of vertex v is computed as concatenating all h_v^n :

$$h_v = \bigoplus_{n \in \{0, \dots, N\}} h_v^n$$

This graph synthesized feature plays an important role in improving the performance of our graph learning algorithms including Covariant Compositional Networks that will be defined in the next chapter. We can also concatenate the dictionary WL and histogram alignment WL graph features into a richer representation.

3.2.3 Dictionary WL graph feature

We combine the Weisfeiler-Lehman graph kernel and Morgan circular fingerprints into the Dictionary Weisfeiler-Lehman graph feature algorithm. To capture the local substructures of a graph, we define a Weisfeiler-Lehman subtree at level/iteration n -th rooted at a vertex v to be the shortest-path subtree that includes all vertices reachable from v by a path of length at most 2^n . Each subtree is represented by a multiset of vertex labels. We build the Weisfeiler-Lehman dictionary by finding all subtree representations of every graph in

the dataset (as in algorithm 6). The graph feature or fingerprint is a frequency vector in which each component corresponds to the frequency of a particular dictionary element (as in algorithm 7).

The remaining question is: How to construct the subtree representation properly? We propose the following solution. Given a multiset of vertex labels M of a subtree, we sort all vertex labels of M in ascending lexicographic order, and concatenate all of them into a string $s(M)$. Finally, we update the dictionary with an element $s(M)$.

One problem of this approach is: Two subtrees with different structures can have the same representing multiset of vertex labels, and is represented by the same element in the dictionary. Definitely we need a more sophisticated representation that is permutation-invariant with respect to the ordering of vertices, while being able to capture the local structures of a graph. This will be fully addressed and discussed in the next section with our proposed model Covariant Compositional Networks.

Regarding data structures to implement our algorithms, we need to have an efficient algorithm for the insertion and searching operations with the dictionary D that can contain millions strings in practice. Our choice is Trie data structure or in another name as Prefix tree. Trie's idea was first introduced by (Briandais, 1959) and then by (Fredkin, 1960). Suppose that our strings only contain ASCII characters indexed from 0 to 255. Each node of the Trie/Prefix tree has exactly 256 pointers, we organize these pointers in an array `next`, each element of `next` corresponds to a character in the ASCII table. There is only a single root node in the Trie/Prefix tree. At the beginning, the Trie only contains the root node with 256 pointers pointing to `NULL`. The insertion operation of a string s to the Trie D can be described as follows. We have a pointer p pointing to the root node at first. We go from

left to right in the string s . For each character $s[i]$, if $p \rightarrow \text{next}[s[i]] = \text{NULL}$ then we allocate a new memory location for $p \rightarrow \text{next}[s[i]]$, and move p to $p \rightarrow \text{next}[s[i]]$. In the end of the insertion algorithm, we mark p as having a string ended at itself or we increase the number of strings ended there. The searching operation can be efficiently implemented in a similar way, except that we do not create any new memory location, but stop searching when we encounter **NULL**.

Algorithm 4 Insertion into Trie

- 1: **Input:** String s and Trie D passed by reference with the **root** node.
- 2: **Output:** Updated Trie D .
- 3: Assign p to **root**
- 4: **for** $i = 1, 2, \dots, |s|$ **do**
- 5: Allocate a new memory location for $p \rightarrow \text{next}[s[i]]$ if it is **NULL**
- 6: Assign p to $p \rightarrow \text{next}[s[i]]$
- 7: **end for**
- 8: Mark p as having a string ended at p , or increase the number of strings ended at p .

Algorithm 5 Searching in Trie

- 1: **Input:** String s and Trie D passed by reference with the **root** node.
- 2: **Output:** Return **true** if found, otherwise return **false**.
- 3: Assign p to **root**
- 4: **for** $i = 1, 2, \dots, |s|$ **do**
- 5: Return **false** immediately if $p \rightarrow \text{next}[s[i]] = \text{NULL}$
- 6: Assign p to $p \rightarrow \text{next}[s[i]]$
- 7: **end for**
- 8: Return **false** if there is no string ended at p , otherwise return **true**.

Suppose that we can access any element of the array of pointers **next** in a constant time $O(1)$. The time complexity of both insertion and searching operations of string s in Trie/Prefix tree is $O(|s|)$ where $|s|$ is length of the string s .

Remark that statements $M \leftarrow M \cup s(S_l(v))$ and $D \leftarrow D \cup s(S_l(v))$ in algorithm 6 will be implemented efficiently with Trie data structure, the time complexity of both statements

Algorithm 6 Finding all dictionary elements representing a graph

- 1: **Input:** Given a graph $G = (V, E)$, label $l(v)$ associated with each vertex $v \in V$, number of iterations T , and the dictionary D passed by reference. Let $S_l(v)$ denote the multiset of labels of the subtree rooted at v in level l -th. Suppose $s(S)$ is a function returns an unique string representing for the multiset of labels S .
- 2: **Output:** Updated dictionary D with new elements found from G , and M is a multiset containing dictionary elements representing G .
- 3: Initialize the WL level 0:
- 4: **for** $v \in V$ **do**
- 5: $S_0(v) \leftarrow \{l(v)\}$
- 6: **end for**
- 7: Build the WL level $1, 2, \dots, T$:
- 8: **for** $l = 1 \rightarrow T$ **do**
- 9: **for** $v \in V$ **do**
- 10: $S_l(v) \leftarrow S_{l-1}(v)$
- 11: **for** $(u, v) \in E$ **do**
- 12: $S_l(v) \leftarrow S_l(v) \cup S_{l-1}(u)$
- 13: **end for**
- 14: **end for**
- 15: **end for**
- 16: Update dictionary D and build the multiset of dictionary elements M :
- 17: $M \leftarrow \emptyset$
- 18: **for** $l = 0 \rightarrow T$ **do**
- 19: **for** $v \in V$ **do**
- 20: $M \leftarrow M \cup s(S_l(v))$
- 21: $D \leftarrow D \cup s(S_l(v))$
- 22: **end for**
- 23: **end for**
- 24: Return D, M .

is $O(|s(S_l(v))|)$. For the sake of simplicity of the complexity analysis, we assume that all set operations (including operations with Trie) can be done in a constant time. The total complexity of both algorithms 6 and 7 is $O(N \cdot T \cdot |E|)$.

Algorithm 7 Building the dictionary for a dataset of graphs

- 1: **Input:** Given a dataset of N graphs $\mathcal{G} = \{G^{(1)}, \dots, G^{(N)}\}$ where $G^{(i)} = (V^{(i)}, E^{(i)})$. Let $l_G : V \rightarrow \Omega$ be the initial feature vector for each vertex of graph $G = (V, E)$. Ω is the set of all possible vertex labels.
 - 2: **Output:** Return the dictionary D and the frequency vector $F^{(i)}$ for each graph $G^{(i)}$. Remark that D is a set of strings, not a multiset. Remark that D is a set of strings, while M is a multiset of strings.
 - 3: Build dictionary D :
 - 4: $D \leftarrow \emptyset$
 - 5: **for** $i = 1 \rightarrow N$ **do**
 - 6: Apply algorithm 6 on graph $G^{(i)}$ to update D and get $M^{(i)}$
 - 7: **end for**
 - 8: Given the dictionary D , build the frequency vectors:
 - 9: **for** $i = 1 \rightarrow N$ **do**
 - 10: $F^{(i)} \leftarrow 0^{|D|}$
 - 11: **for** $m \in M^{(i)}$ **do**
 - 12: Search for m in D , let say $m = D_j$
 - 13: $F_j^{(i)} \leftarrow F_j^{(i)} + 1$
 - 14: **end for**
 - 15: $F^{(i)} \leftarrow F^{(i)} / \|F^{(i)}\|$ where $\|\cdot\|$ denotes the norm l_2 of the vector
 - 16: **end for**
 - 17: Return $D, \{F^{(1)}, \dots, F^{(N)}\}$
-

3.2.4 Shortest path graph kernel

Before defining the shortest path kernel, we start with all-paths kernel suggested by (Borgwardt and Kriegel, 2005).

Definition 3.2.2 (All-paths kernel). Given two graphs G_1 and G_2 . Let $P(G_i)$ be the set of all paths in graph G_i where $i \in \{1, 2\}$. Let k_{path} be a positive kernel on two graphs, defined as the product of kernels on edges and nodes along the paths. We then define an all-paths

kernel $k_{all\ paths}$ as

$$k_{all\ paths}(G_1, G_2) \triangleq \sum_{p_1 \in P(G_1)} \sum_{p_2 \in P(G_2)} k_{path}(p_1, p_2)$$

In other words, we define the all-paths kernel as the sum over all kernels on pairs of paths from G_1 and G_2 .

Lemma 3.2.1. *The all-paths kernel is positive definite.*

Proof. Detail of the proof is contained in (Borgwardt and Kriegel, 2005). \square

Lemma 3.2.2. *Computing the all-paths kernel is NP-hard.*

Proof. Suppose that determining the set of all paths $P(G)$ in a graph $G = (V, E)$ is not NP-hard. There exists a polynomial time (of $|V|$) algorithm to determine whether G has a Hamilton path by checking if $P(G)$ contains a path of length $|V| - 1$. However, this problem is known to be NP-complete. Therefore, determining the set of all paths and computing the all-paths kernel are NP-hard problems. \square

From Lemma 3.2.2, we conclude that the all-paths kernel is infeasible for computation. Thus, as following, we consider the shortest-path kernel. First, for the sake of mathematical convenience to define the shortest-path kernel, we introduce the Floyd-transformed graph.

Definition 3.2.3. Given an undirected connected weighted graph $G = (V, E)$, the Floyd-transformed graph of G is a complete graph $S = (V, \overline{E})$ in which for all $(u, v) \in \overline{E}$, the weight of edge (u, v) is length of the shortest-path between u and v in G .

We can easily construct the Floyd-transformed graph S of G by Floyd-Warshall algorithm in $O(|V|^3)$. After Floyd-transformation of the input graphs, a definition of shortest-path kernel is introduced as follows.

Definition 3.2.4 (Shortest-path graph kernel). Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs that are Floyd-transformed into $S_1 = (V_1, \bar{E}_1)$ and $S_2 = (V_2, \bar{E}_2)$. The shortest-path graph kernel is defined as

$$k_{\text{shortest paths}}(G_1, G_2) \triangleq \sum_{e_1 \in \bar{E}_1} \sum_{e_2 \in \bar{E}_2} k_{\text{walk}}^{(1)}(e_1, e_2)$$

where $k_{\text{walk}}^{(1)}$ is a positive definite kernel on edge walks of length 1.

Label enrichment can also be applied to Floyd transformed graphs to speedup kernel computation. When performing the Floyd-Warshall algorithm, besides storing the shortest path information for each pair of vertices, we also store the number of edges on the shortest path. The equal length shortest-path kernel can be done by setting kernels to zero for all pairs of shortest paths where the number of edges in the shortest paths is not identical (Borgwardt and Kriegel, 2005). Regarding the dynamic programming (DP) algorithm, let $d_{u,v}$ and $d_{u,v}^{(e)}$ denote lengths of the shortest path and the shortest path with exactly e edges between $u, v \in V$. The DP formula of Floyd-Warshall is

$$d_{u,v} = \min_{k \in V} \{d_{u,k} + d_{k,v}\}$$

while the DP formula of Floyd-Warshall with number of edges information is

$$d_{u,v}^{(e)} = \min_{k \in V, e_1 + e_2 = e} \{d_{u,k}^{(e_1)} + d_{k,v}^{(e_2)}\}$$

For a given e , we construct the corresponding Floyd transformed graph of shortest paths with length exactly e edges. From here, we have our definition of Equal-length Shortest-path (ELSP) kernel.

Definition 3.2.5 (Equal-length Shortest-path kernel). Given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. For a non-zero natural number $e \in [1, \min(|V_1|, |V_2|) - 1]$, we construct the

Floyd transformed graphs of e -edge shortest paths $S_1 = (V_1, E_1^{(e)})$ and $S_2 = (V_2, E_2^{(e)})$ for G_1 and G_2 , respectively. The equal-length shortest-path kernel is defined as

$$k_{ELSP}(G_1, G_2) \triangleq \sum_{e=1}^{\min(|V_1|, |V_2|)-1} \sum_{e_1 \in E_1^{(e)}} \sum_{e_2 \in E_2^{(e)}} k_{walk}^{(1)}(e_1, e_2)$$

3.2.5 Graphlet kernel

State-of-the-art graph kernels do not scale to large graphs with hundreds of vertices and thousands of edges. To address this issue, (Shervashidze et al., 2009) proposed Graphlet kernel exploiting frequent subgraph mining algorithms that aims to detect subgraphs that are frequent in a given dataset of graphs. Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* (denoted by $G \cong G'$) if there exists a bijective mapping $f : V \rightarrow V'$ (called the isomorphism function) such that $(v_i, v_j) \in E$ if and only if $(f(v_i), f(v_j)) \in E'$. In graph theory, graphlet is a small, connected, non-isomorphic, induced subgraph (that must contain all edges between its vertices) of a given graph. Let $\mathcal{G}_n = \{g(1), \dots, g(N_n)\}$ be the set of size- n graphlets. It is trivial to see that:

- $n = 1$: $N_1 = 1$ and \mathcal{G}_1 contains only 1 graph that has 1 vertex.
- $n = 2$: $N_2 = 2$ and \mathcal{G}_2 contains 2 graphs of 2 vertices, one graph has no edge, and the another one has 1 edge.
- $n = 3$: $N_3 = 4$, there are 4 non-isomorphic graphs with 3 vertices.
- $n = 4$: $N_4 = 11$, there are 11 non-isomorphic graphs with 4 vertices.

Given a graph G , define a vector $f_G^{(n)}$ of length N_n whose i -th component corresponds to the frequency of occurrence of $g(i)$ in G . We will call $f_G^{(n)}$ the n -spectrum of G . This statistic is the foundation of our novel graph kernel.

Definition 3.2.6 (Graphlet kernel). Given two graphs G and G' of size greater than n , the graphlet kernel of graphlet size n is defined as:

$$k_{\text{graphlet}}^{(n)}(G, G') \triangleq \langle f_G^{(n)}, f_{G'}^{(n)} \rangle$$

In order to account for differences in the sizes of the graphs, we normalize the frequency counts $f_G^{(n)}$ to probability vectors:

$$k_{\text{graphlet}}^{(n)}(G, G') \triangleq \frac{\langle f_G^{(n)}, f_{G'}^{(n)} \rangle}{\|f_G^{(n)}\|_1 \cdot \|f_{G'}^{(n)}\|_1}$$

In the precomputation step, we need to find the set \mathcal{G}_n of N_n non-isomorphic graphs of size n . There are $\binom{|V|}{n}$ size- n subgraphs in a graph G , computing $f_G^{(n)}$ requires $O(|V|^n)$. For each subgraph of G , we need to classify it into 1 element of \mathcal{G}_n by a graph isomorphism test that is well-known to be a NP-complete problem. Therefore, the computation time for graphlet kernel with large n is still extremely expensive.

3.2.6 Random walk graph kernel

Generalized random walk graph kernels are based on a simple idea: given a pair of graphs, perform random walks on both, and count the number of matching walks (Vishwanathan et al., 2010). First, we construct the direct product graph of two graphs.

Definition 3.2.7 (Kronecker product). Given real matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{p \times q}$, the Kronecker product $A \otimes B \in \mathbb{R}^{np \times mq}$ is defined as:

$$A \otimes B \triangleq \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1m}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B & A_{n2}B & \dots & A_{nm}B \end{bmatrix}$$

Definition 3.2.8 (Direct product graph). Given two graphs $G = (V, E)$ and $G' = (V', E')$ with $|V| = n$ and $|V'| = n'$, their direct product G_{\times} is a graph with vertex set

$$V_{\times} = \{(v_i, v'_{i'}) : v_i \in V, v'_{i'} \in V'\}$$

and edge set

$$E_{\times} = \{((v_i, v'_{i'}), (v_j, v'_{j'})) : (v_i, v_j) \in E \wedge (v'_{i'}, v'_{j'}) \in E'\}$$

If A and A' are the respective adjacency matrices of G and G' , then the adjacency matrix of G_{\times} is $A_{\times} = A \otimes A'$.

In other words, G_{\times} is a graph over pairs of vertices from G and G' , and two vertices in G_{\times} are neighbors if and only if the corresponding vertices in G and G' are both neighbors. Performing a random walk on the direct product graph is equivalent to performing a simultaneous random walk on G and G' . Let p and p' denote the starting probability distributions over the vertices of G and G' . Let q and q' denote the stopping probability distributions over the vertices of G and G' . The corresponding starting and stopping probabilities on the direct product graph are $p_{\times} = p \otimes p'$ and $q_{\times} = q \otimes q'$, respectively. Let $|V| = n$ and $|V'| = n'$. If G and G' are edge-labeled (discrete labels), we can associate a weight matrix $W_{\times} \in \mathbb{R}^{nn' \times nn'}$ with G_{\times} :

$$W_{\times} = \sum_{\ell=1}^d A^{\ell} \otimes A'^{\ell} \tag{3.9}$$

where d is the number of different labels, A^{ℓ} and A'^{ℓ} are the filtered adjacency matrices of G and G' by label ℓ (we keep the edge weights of edges with label ℓ , and set edge-weight 0 to all other edges). In the case the graphs are unlabeled, we can set $W_{\times} = A_{\times}$. Let A_{\times}^k be the probability of simultaneous length k random walks on G and G' . Let W_{\times}^k be the similarity between simultaneous length k random walks on G and G' , measured via a kernel function \mathcal{K} . Given initial and stopping probability distributions p_{\times} and q_{\times} one can compute $q_{\times}^T W_{\times}^k p_{\times}$, which is expected similarity between simultaneous length k random walks on G and G' .

To define a kernel which computes the similarity between G and G' , one natural idea is to simply sum up $q_{\times}^T W_{\times}^k p_{\times}$ for all values of k . To achieve the convergence, we introduce appropriate chosen non-negative coefficients $\mu(k)$ in the definition of kernel between G and G' :

$$k(G, G') \triangleq \sum_{k=0}^{\infty} \mu(k) q_{\times}^T W_{\times}^k p_{\times} \quad (3.10)$$

Based on the generic equation 3.10 of random walk kernel, several authors have defined special cases in the literature.

Definition 3.2.9 (Special case 1 - (Kashima et al., 2004)). Assume that $\mu(k) = \lambda^k$ for some $\lambda > 0$, we can write:

$$k(G, G') \triangleq \sum_{k=0}^{\infty} \lambda^k q_{\times}^k W_{\times}^k p_{\times} = q_{\times}^T (I - \lambda W_{\times})^{-1} p_{\times}$$

Definition 3.2.10 (Special case 2 - (Gärtner et al., 2003)). Assuming uniform distributions for the starting and stopping probabilities over the vertices of G and G' , the kernel can be defined as counting the number of matching walks:

$$k(G, G') \triangleq \frac{1}{n^2 n'^2} \sum_{i=1}^n \sum_{j=1}^{n'} \sum_{k=0}^{\infty} \mu(k) [A_{\times}^k]_{ij}$$

Definition 3.2.11 (Special case 3 - (Vishwanathan, 2002)). *Exponential* random walk kernel can be defined as:

$$k(G, G') \triangleq \sum_{i=1}^n \sum_{j=1}^{n'} [e^{\lambda A_{\times}}]_{ij}$$

3.2.7 Laplacian graph kernels

The first Laplacian graph kernel was originally introduced by (Johansson and Dubhashi, 2015) and (Kondor and Pan, 2016) proposed its generalization to capture multiscale and

hierarchical structures. In either case, the graph Laplacian is used for constructing a similarity function on graphs. Recall that the graph Laplacian of weighted undirected graph $G = (V, E)$ is an $|V| \times |V|$ positive semi-definite matrix L , with

$$L_{i,j} = \begin{cases} -w_{i,j} & \text{if } \{v_i, v_j\} \in E \\ \sum_{j:(v_i, v_j) \in E} w_{i,j} & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases} \quad (3.11)$$

where $w_{i,j}$ is the weight of edge (v_i, v_j) . Given the weighted degree matrix D , Eq. 3.11 can be expressed as $L = D - A$. Given two graphs G_1 and G_2 of the same number nodes n , with corresponding graph Laplacians L_1 and L_2 , respectively. We can define the kernel between them to be a kernel between two Normal distributions $p_1 = \mathcal{N}(0, L_1^{-1})$ and $p_2 = \mathcal{N}(0, L_2^{-1})$. One option is the Bhattacharyya kernel (Jebara and Kondor, 2003),

$$k(p_1, p_2) = \int \sqrt{p_1(x)} \sqrt{p_2(x)} dx,$$

that can be written in a closed form for Gaussian distributions,

$$k(p_1, p_2) = \frac{|(\frac{1}{2}L_1 + \frac{1}{2}L_2)^{-1}|^{1/2}}{|L_1^{-1}|^{1/4} |L_2^{-1}|^{1/4}}.$$

If some of the eigenvalues of L_1^{-1} and L_2^{-1} are zero or very close of zero, we might encounter vanishingly small kernel values. Thus, (Kondor and Jebara, 2003) proposed the regularization by adding some small constant γ times the identity to L_1^{-1} and L_2^{-1} , that leads to the construction of the positive semi-definite Laplacian graph (LG) kernel based on the Bhattacharyya kernel,

$$k_{LG}(G_1, G_2) = \frac{|(\frac{1}{2}S_1 + \frac{1}{2}S_2)^{-1}|^{1/2}}{|S_1^{-1}|^{1/4} |S_2^{-1}|^{1/4}}, \quad (3.12)$$

where $S_1 = L_1^{-1} + \gamma I$ and $S_2 = L_2^{-1} + \gamma I$. One way to interpret the kernel construction in Eq. 3.12 is via graphical model such that we use G to construct a pairwise Markov Random Field (MRF)¹ over n variables x_1, \dots, x_n with edge potentials and node potentials,

$$\phi(x_i, x_j) = e^{-w_{i,j}(x_i - x_j)^2/2}, \quad \psi(x_i) = e^{-\gamma x_i^2/2},$$

respectively. The joint distribution of $\mathbf{x} = (x_1, \dots, x_n)^T$ is then

$$p(\mathbf{x}) \propto \prod_{(v_i, v_j) \in E} \phi(x_i, x_j) \prod_{v_i \in V} \psi(x_i) = e^{-\mathbf{x}^T (L + \gamma I) \mathbf{x}/2},$$

where the covariance matrix of \mathbf{x} is $(L + \gamma I)^{-1}$ and γ acts as a small constant regularizer. Some limitations of the LG kernel are:

1. It only works in the case two graphs have the same number of nodes,
2. Spectral graph theory suggests that the eigenvectors corresponding to the low eigenvalues of L are informative about the overall shape of the graph (see appendix section 10.3). Thus, it is only sensitive to the overall structure of the two graphs.
3. It is not permutation-invariant.

To obtain a permutation-invariant kernel, (Kondor and Pan, 2016) suggests the use of node features. Suppose we are given (f_1, \dots, f_m) as a collection of m local node features. Then we define the corresponding feature mapping matrices $[U_1]_{i,j} = \phi_i(v_j)$ and $[U_2]_{i,j} = \phi_i(v'_j)$ where v_j and v'_j are the j -th nodes of G_1 and G_2 , respectively. The feature space Laplacian graph kernel (FLG) is defined the same as Eq. 3.12 but with $S_1 = U_1 L_1^{-1} U_1^T + \gamma I$ and $S_2 = U_2 L_2^{-1} U_2^T + \gamma I$. Because the f_1, \dots, f_m are node features and invariant to node permutation, the

1. We call it pairwise MRF in the sense that we only consider maximal cliques of size at most 2 in which a single node is a maximal clique of size 1 and an edge is a maximal clique of size 2. The general form with all-size maximal cliques can be obtained from the Hammersley-Clifford theorem.

resulted FLG kernel is indeed permutation-invariant. The multiscale extension, proposed by (Kondor and Pan, 2016), allows us to not just capture the topological relationships between individual vertices, but also the topological relationships between subgraphs. The recursive construction and the analysis of the multiscale graph Laplacian kernel are detailed in the original work of (Kondor and Pan, 2016).

3.2.8 Diffusion kernels

In this section, we will discuss about the well-known *diffusion kernel* (Kondor and Lafferty, 2002). However, the setting here is different from all other graph kernels we discussed above. Before, we designed positive semi-definite kernels $k(G_1, G_2)$ measuring the similarity between two graphs G_1 and G_2 . But in this case, fundamentally we design a kernel $k(v, v')$ measuring the similarity between two nodes v and v' of a single graph G , given the inspiration from the differential equation describing heat conduction.

First, to understand the diffusion kernel, we recall the definition of the *matrix exponential* of a matrix L that is analogous with the exponentiation of real numbers:

$$e^{\beta L} \triangleq \lim_{s \rightarrow \infty} \left(I + \frac{\beta L}{s} \right)^s, \quad s \in \mathbb{N}.$$

Matrix exponentiation shares many properties with the ordinary exponential function, including the power series expansion

$$e^{\beta L} = I + \beta L + \frac{\beta^2}{2} L^2 + \frac{\beta^3}{6} L^3 + \dots$$

There are few differences comparing to the ordinary exponentiation:

- $e^{\beta L}$ yields a matrix, but is not equivalent to componentwise exponentiation $e^{\beta L_{ij}}$.
- For the matrix case, $e^{\beta(A+B)} \neq e^{\beta A} e^{\beta B}$.

It is important to note that we can get a positive definite kernel, called *exponential kernel*, by simply writing:

$$e^{\beta L} = \lim_{s \rightarrow \infty} \left(I + \frac{\beta L}{s} \right)^s = \lim_{2s \rightarrow \infty} \left(I + \frac{\beta L}{2s} \right)^{2s},$$

where L is a symmetric matrix, because of the fact that any even power of a symmetric matrix is always positive definite.

Inspired by the diffusion equation and the exponential kernel, (Kondor and Lafferty, 2002) defines the *diffusion kernel* or *heat kernel* on a graph G as the limit

$$K_\beta \triangleq e^{\beta L} \lim_{s \rightarrow \infty} \left(I + \frac{\beta L}{s} \right)^s, \quad s \in \mathbb{N}, \quad (3.13)$$

where $\beta \in \mathbb{R}$ is a real constant, I is the identity matrix and L is the graph Laplacian. The formula 3.13 corresponds to a random walk with an infinite number of infinitesimally small steps and the time evolution of this continuous-time random walk is governed by the graph Laplacian L . Differentiating Eq.3.13 with respect to β results into a differential equation

$$\frac{d}{d\beta} K_\beta = L K_\beta, \quad (3.14)$$

with the accompanying initial conditions $K(0) = I$. It is natural to interpret that $K(\beta)$ is the product of a continuous process, expressed by L , gradually transforming it from the identity matrix (i.e. $K(0)$) to a kernel with stronger and stronger off-diagonal effects as β increases. Indeed, this is the actual physical processes of diffusion and Eq. 3.14 is referred as the *heat equation* or *diffusion equation*. Furthermore, the parameter β controls the extent of the diffusion, or to specify the length scale, similarly to σ in the Gaussian kernel,

$$k(x, x') = \frac{1}{(2\pi\sigma^2)^{d/2}} e^{-\|x-x'\|^2/(2\sigma^2)}.$$

Fixing $x' = x_0$ and letting $t = \sigma^2/2$, the Gaussian kernel becomes

$$k_{x_0}(x, t) = \frac{1}{(4\pi t)^{d/2}} e^{-\|x-x_0\|^2/(4t)},$$

that is the solution of the diffusion equation,

$$\frac{\partial}{\partial t} k_{x_0}(x, t) = \left[\frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \cdots + \frac{\partial^2}{\partial x_d^2} \right] k_{x_0}(x, t), \quad (3.15)$$

with Dirac delta initial condition $k_{x_0}(x, 0) = \delta(x - x_0)$. We define the Laplacian operator as

$$\Delta = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \cdots + \frac{\partial^2}{\partial x_d^2}. \quad (3.16)$$

The Eq. 3.15 is then reduced to

$$\frac{\partial}{\partial t} k_{x_0}(x, t) = \Delta k_{x_0}(x, t), \quad (3.17)$$

that describes how heat and gases, introduced at x_0 , diffuse with time in a homogeneous, isotropic medium. Fundamentally, the two diffusion equations at 3.14 and 3.17 are the same: L is the Laplacian operator on graph, while Δ is the continuous Laplacian operator. One can show that if we discretize the domain as a two-dimensional grid graph, the graph Laplacian is an approximation for the continuous one (see appendix section 10.3).

The last question is how to efficiently compute the diffusion kernel as directly applying the Def. 3.13 results in so many matrix multiplications. First, we compute the normalized eigenvectors u_1, u_2, \dots, u_n and the corresponding eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$ of the graph Laplacian L . By orthogonality, we can write

$$L^2 = \left(\sum_{i=1}^n u_i \lambda_i u_i^T \right)^s = \sum_{i=1}^n u_i \lambda_i^s u_i^T,$$

from which

$$e^{\beta L} = I + \left(\sum_{i=1}^n u_i \beta \lambda_i u_i^T \right) + \left(\sum_{i=1}^n u_i \frac{(\beta \lambda_i)^2}{2} u_i^T \right) + \dots = \sum_{i=1}^n u_i e^{\beta \lambda_i} u_i^T.$$

Thus, we reduce from matrix exponentiation $e^{\beta L}$ to real exponentiation $e^{\beta \lambda_i}$. The complexity of diagonalizing the graph Laplacian is $O(n^3)$, and so is the complexity of the computing the diffusion kernel. The diffusion kernel on graph can be generalized to Riemannian manifolds as proposed in (Lafferty and Lebanon, 2005).

3.3 Using graph kernels

So far in this chapter, we have discussed several graph kernels that can thought as a way of comparing the similarity and difference between two graphs. In this section, we will discuss about how to use a kernel (e.g., a graph kernel) for the downstream tasks such as regression (e.g., estimating molecular properties) and classification (e.g., predicting discrete graph labels). The most widely used and famous methods are Gaussian Processes (GP) and Support Vector Machine (SVM).

3.3.1 Gaussian Processes

In the setting of supervised learning, we observe some inputs x_i and some outputs y_i . In the case we are interested in, each sample x_i is indeed a graph and $y_i \in \mathbb{R}$ is some real-valued graph label we are trying to regress. We assume that $y_i = f(x_i)$ for some unknown function f that can be corrupted by noise. We denote \mathbf{X} and \mathbf{y} as the set of all points and all labels, respectively. The data is denoted by $\mathcal{D} = (\mathbf{X}, \mathbf{y})$. The optimal approach is to infer a *distribution over functions* given the data, $p(f|\mathbf{X}, \mathbf{y})$, and then to use this to make

prediction y_* given a new input x_* , i.e., to compute

$$p(y_*|x_*, \mathbf{X}, \mathbf{y}) = \int p(y_*|f, x_*)p(f|\mathbf{X}, \mathbf{y})df,$$

that we need a way to perform Bayesian inference over functions themselves. This is fundamentally different from parameter inference in which we have parametric representations $\boldsymbol{\theta}$ for the function f and we infer $p(\boldsymbol{\theta}|\mathcal{D})$. Instead, we infer $p(f|\mathcal{D})$ directly. One of the ways is Gaussian processes that defines a prior over functions, which can be converted into a posterior over functions given the data. Given a finite arbitrary set of points $\{x_1, \dots, x_n\}$, a GP assumes that $p(f(x_1), \dots, f(x_n))$ is jointly Gaussian, with some mean $\mu(x)$ and covariance $\Sigma(x)$ given by $\Sigma_{i,j} = k(x_i, x_j)$, where $k(\cdot, \cdot)$ is a positive definite kernel function (e.g., one of any graph kernels we have defined previously). The intuition behind this setup is that if x_i and x_j are similar as suggested by a small distance $k(x_i, x_j)$, then we expect the corresponding outputs y_i and y_j tend to be close as well.

Let the prior on the regression function be a GP, denoted by

$$f(x) \sim \text{GP}(\mu(x), k(x, x')),$$

where $\mu(x)$ is the mean function and $k(x, x')$ is the kernel or covariance function, i.e.,

$$\mu(x) = \mathbb{E}[f(x)], \quad k(x, x') = \mathbb{E}[(f(x) - \mu(x))(f(x') - \mu(x'))^T].$$

For any finite set of points, this process defines a joint Gaussian:

$$p(\mathbf{y}|\mathbf{X}) = \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where $\Sigma_{i,j} = k(x_i, x_j)$ and $\boldsymbol{\mu} = (\mu(x_1), \dots, \mu(x_n))$. It is known that GP is flexible enough

to model the mean arbitrarily well, so using zero mean function, i.e. $\mu(x) = 0$, is indeed a common practice.

First, we consider how GP makes predictions using *noise-free* observations. Suppose we observe a training set of n examples $\mathcal{D} = (\mathbf{X}, \mathbf{y}) = \{(x_i, y_i)\}_{i=1}^n$, where $y_i = f(x_i)$ is the noise-free (i.e. zero noise) observation of the function evaluated at x_i . Furthermore, suppose that we are given a test set \mathbf{X}_* of n_* points, that is stored as a matrix of size $n_* \times d$ where d is the number of input dimensions. We want to predict the function outputs y_* (i.e. as a vector of n_* elements). Given the fact that observations are *noiseless*, we want the GP to act as an *interpolator* of the training data, i.e. the GP returns the answer $f(x_i)$ with no uncertainty. By definition, the joint distribution has the following form

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{y}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right), \quad (3.18)$$

where $\mathbf{K} = k(\mathbf{X}, \mathbf{X})$ is the $n \times n$ kernel gram-matrix on the training set \mathcal{D} , $\mathbf{K}_* = k(\mathbf{X}, \mathbf{X}_*)$ is the $n \times n_*$ kernel matrix between the training points \mathbf{X} and the testing points \mathbf{X}_* , and $\mathbf{K}_{**} = k(\mathbf{X}_*, \mathbf{X}_*)$ is the $n_* \times n_*$ kernel gram-matrix on the testing set.

Theorem 3.3.1 (Marginals and conditionals of an MVN). *Suppose $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ is jointly Gaussian with parameters*

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}, \quad \Lambda = \Sigma^{-1} = \begin{pmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{pmatrix}.$$

Then the marginals are given by

$$p(\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_1, \Sigma_{11})$$

$$p(\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_2 | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22})$$

and the posterior conditional is given by $p(\mathbf{x}_1 | \mathbf{x}_2) = \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2})$ with parameters

$$\boldsymbol{\Sigma}_{1|2} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1},$$

and

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2) = \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\mathbf{x}_2 - \boldsymbol{\mu}_2)$$

that is also equivalent to

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\Sigma}_{1|2}(\boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12}(\mathbf{x}_2 - \boldsymbol{\mu}_2)).$$

Theorem 3.3.1 gives us a way to compute the conditionals $p(\mathbf{x}_1 | \mathbf{x}_2)$ given a joint distribution $p(\mathbf{x}_1, \mathbf{x}_2)$ for the case of multivariate normal (MVN) distributions. Applying the theorem, from Eq. 3.18, the posterior has the following form $p(\mathbf{y}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{y}_* | \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ with parameters

$$\boldsymbol{\mu}_* = \boldsymbol{\mu}(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1}(\mathbf{y} - \boldsymbol{\mu}(\mathbf{X}))$$

and

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_*.$$

The time complexity of the computation is $O(d^3)$ in the worst case.

In the case we are given *noisy* observations, i.e. $y = f(x) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ that we assume to be a white noise (i.e. noise with zero mean), GP does not have to interpolate the data. The covariance of the observed noisy responses is

$$\text{cov}[y_i, y_j] = k(x_i, x_j) + \sigma^2 \delta_{ij},$$

where $\delta_{ij} = \mathbb{I}[i = j]$ denotes the Kronecker delta. In other words,

$$\text{cov}[\mathbf{y}|\mathbf{X}] = \mathbf{K} + \sigma^2 \mathbf{I} \triangleq \mathbf{K}_\sigma.$$

Similar to 3.18, the joint density of the observed data and the noise-free function on the test points is given by

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{y}_* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K}_\sigma & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right),$$

where we are assuming the mean is zero. Therefore, the posterior predictive density $p(\mathbf{y}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{y}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ has parameters

$$\boldsymbol{\mu}_* = \mathbf{K}_*^T \mathbf{K}_\sigma^{-1} \mathbf{y} = \mathbf{K}_*^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y}$$

and

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}_\sigma^{-1} \mathbf{K}_* = \mathbf{K}_{**} - \mathbf{K}_*^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{K}_*.$$

In practice, σ is usually treated as a hyperparameter that needs tuning. If we only have a single test input x_* , we have

$$p(y_*|x_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(y_*|\mathbf{k}_*^T \mathbf{K}_\sigma^{-1} \mathbf{y}, k_{**} - \mathbf{k}_*^T \mathbf{K}_\sigma^{-1} \mathbf{k}_*),$$

where $\mathbf{k}_* = [k(x_*, x_1), \dots, k(x_*, x_n)]^T$ and $k_{**} = k(x_*, x_*)$. The posterior mean can be also written as

$$\mu_* = \mathbf{k}_*^T \mathbf{K}_\sigma^{-1} \mathbf{y} = \sum_{i=1}^n \alpha_i k(x_i, x_*), \quad (3.19)$$

where $\boldsymbol{\alpha} = \mathbf{K}_\sigma^{-1} \mathbf{y}$. The expression in Eq. 3.19 is indeed important and related to the other statistical machine learning methods including SVM, that we will discuss further.

3.3.2 Kernelized ridge regression

Previously, we have discussed Gaussian Processes as a method of Bayesian inference on functions, and how to use our kernels to make predictions. Now, we will discuss the another approach, parameter inference, in which we represent the function we want to estimate with parameters θ and infer $p(\theta|\mathcal{D})$ from the observed data \mathcal{D} . It is not so trivial how to apply kernels including graph kernels to parametric models such as ridge regression. However, it can be done as explained below and kernelized ridge regression is a useful method for some of our graph learning experiments in this chapter.

First, we discuss the primal problem. For the case of vectorized inputs, let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote the design matrix and let $y \in \mathbb{R}^n$ denote the corresponding labels. We want to minimize the following loss with parameter regularization,

$$\mathcal{L}(\theta) = \|y - \mathbf{X}\theta\|^2 + \lambda\|\theta\|^2 = (y - \mathbf{X}\theta)^T(y - \mathbf{X}\theta) + \lambda\|\theta\|^2.$$

The optimal solution is given by

$$\theta = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T y = \left(\sum_{i=1}^n x_i x_i^T + \lambda \mathbf{I} \right)^{-1} \mathbf{X}^T y,$$

that by using the matrix inversion lemma, we can further rewrite as

$$\theta = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} y,$$

which takes $O(n^3 + n^2d)$ time complexity to compute (e.g., that can be advantageous if d is large).

Now, we discuss the dual problem. The term $\mathbf{X} \mathbf{X}^T$ can be seen as the gram matrix of the

linear kernel. We replace this term by the kernel matrix \mathbf{K} of some other kernel functions (e.g., our graph kernels). We define the *dual variables* as

$$\alpha \triangleq (\mathbf{K} + \lambda \mathbf{I})^{-1} y.$$

We can rewrite the *primal variables* as

$$\theta = \mathbf{X}^T \alpha = \sum_{i=1}^n \alpha_i x_i. \quad (3.20)$$

In the case of vectorized data, this expression tells us that the solution vector θ is just a linear sum of the n training vectors. When we plug this in at test time to compute the predictive mean, we get

$$\hat{f}(x) = \theta^T x = \sum_{i=1}^n \alpha_i x_i^T x = \sum_{i=1}^n \alpha_i k(x, x_i).$$

Importantly, this kernelized expression is indeed similar to Eq. 3.19, that allows us to apply ridge regression on graph data via graph kernels.

3.3.3 Support Vector Machines

The problem with kernelized ridge regression is that the solution vector θ depends on all the training inputs as shown in Eq. 3.20. Our motivation is to seek a method to produce a *sparse* estimate. In other words, we want to ensure that the solution is sparse, i.e. the predictions only depend on a subset of the training data, called as *support vectors*. (Vapnik et al., 1996) originally proposed *Support Vector Machine* (SVM) as a combination of the kernel trick and modified loss function for binary classification, but can be extended to regression and multi-class classification.

SVMs for regression

Consider the ℓ_2 regularized empirical risk function

$$\mathcal{L}(\theta, \lambda) = \sum_{i=1}^n L(y_i, \hat{y}_i) + \lambda \|\theta\|^2,$$

where $\hat{y}_i = \hat{f}(x_i) = \theta_0 + \theta^T x_i$ is the prediction at point x_i . Here we denote $\theta_0 \in \mathbb{R}$ as the bias term, separately from vector $\theta \in \mathbb{R}^d$; and for convenience, we denote both of them as $\boldsymbol{\theta}$. If L is quadratic loss, this is equivalent to ridge regression; and if L is the log-loss, this is equivalent to logistic regression. For SVM, (Vapnik et al., 1996) proposed a variant of the Huber loss function called the *epsilon insensitive loss function*, defined by

$$L_\epsilon \triangleq \begin{cases} 0 & \text{if } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise} \end{cases}.$$

This means that any prediction lying inside an ϵ -interval around the ground-truth is not penalized. The corresponding objective function is usually written as

$$\mathcal{L}(\boldsymbol{\theta}, C) = C \sum_{i=1}^n L_\epsilon(y_i, \hat{y}_i) + \frac{1}{2} \|\theta\|^2,$$

where $C = \frac{1}{\lambda}$ is a regularization constant. This objective is convex, but not differentiable (because of the absolute value function in the loss term), and unconstrained. We formulate the problem as a constrained optimization by introducing the *slack variables* $\boldsymbol{\xi} = \{\xi_i^+, \xi_i^-\}_{i=1}^n$ to represent the degree to which each point lies outside the ϵ -interval such that $y_i \leq \hat{f}(x_i) + \epsilon + \xi_i^+$ and $y_i \geq \hat{f}(x_i) - \epsilon - \xi_i^-$. We have the following constrained optimization

$$\min_{\boldsymbol{\theta}, \boldsymbol{\xi}} C \sum_{i=1}^n (\xi_i^+ + \xi_i^-) + \frac{1}{2} \|\theta\|^2$$

subject to, for all $i \in \{1, \dots, n\}$:

$$y_i \leq \hat{f}(x_i) + \epsilon + \xi_i^+,$$

$$y_i \geq \hat{f}(x_i) - \epsilon - \xi_i^-,$$

$$\xi_i^+, \xi_i^- \geq 0.$$

This is a standard quadratic program with $2n + d + 1$ variables. (Schölkopf and Smola, 2002) shows that the optimal solution has the form

$$\hat{\theta} = \sum_{i=1}^n \alpha_i x_i$$

where $\alpha_i \geq 0$. Furthermore, α is a sparse vector, because the errors which are smaller than ϵ are neglected. The points x_i for which $\alpha_i > 0$ are called the *support vectors* – points for which the errors lie on or outside the ϵ -interval. Once the model is trained, for the case of using linear kernel, we can make the predictions using:

$$\hat{y}(x) = \hat{\theta}_0 + \theta^T x = \hat{\theta}_0 + \sum_{\alpha_i > 0} \alpha_i x_i^T x;$$

and for the case of using a customized kernel $k(\cdot, \cdot)$:

$$\hat{y}(x) = \hat{\theta}_0 + \sum_{\alpha_i > 0} \alpha_i k(x_i, x).$$

SVMs for classification

First, we focus on the case of binary classification $y \in \{+1, -1\}$, and then the multi-class case can be simply derived. In a logistic regression model, we use the loss as the negative log likelihood (NLL) as

$$L_{\text{nll}}(y, \eta) = -\log p(y|x, \theta) = \log(1 + e^{-y\eta}),$$

where $\eta = f_\theta(x) = \theta_0 + \theta^T x$. For SVM, we replace the NLL loss with the *hinge loss*, defined

as

$$L_{\text{hinge}}(y, \eta) = \max(0, 1 - y\eta) = (1 - y\eta)_+.$$

The overall objective has the form

$$\min_{\theta} C \sum_{i=1}^n (1 - y_i f_{\theta}(x_i))_+ + \frac{1}{2} \|\theta\|^2,$$

that is non-differentiable, because of the max term. By introducing slack variables $\xi = \{\xi_i\}_{i=1}^n$, this is equivalent to solving

$$\min_{\theta, \xi} \mathcal{L}_{\text{primal}}(\theta, \xi) \triangleq C \sum_{i=1}^n \xi_i + \frac{1}{2} \|\theta\|^2 \quad (3.21)$$

subject to, for all $i \in \{1, \dots, n\}$:

$$y_i(\theta_0 + \theta^T x_i) \geq 1 - \xi_i, \quad (3.22)$$

$$\xi_i \geq 0. \quad (3.23)$$

This is a quadratic program in $n + d + 1$ variables, subject to $O(n)$ constraints. It can be shown that the solution has the form $\hat{\theta} = \sum_{i=1}^n \alpha_i y_i x_i$ and α is sparse. Later we will see that α_i plays the role of a Lagrange multiplier. The points x_i for which $\alpha_i > 0$ are called support vectors, which are either incorrectly classified, or are classified correctly but are on or inside the margin. At test time, for linear kernel, the prediction is

$$\hat{y}(x) = \text{sign}(f_{\hat{\theta}}(x)) = \text{sign}(\hat{\theta}_0 + \hat{\theta}^T x_i);$$

and for arbitrary kernel:

$$\hat{y}(x) = \text{sign}\left(\hat{\theta}_0 + \sum_{i=1}^n \alpha_i k(x_i, x)\right).$$

The computation is proportional to the number of support vectors that depends on the sparsity level, and hence on the regularizer C .

Indeed, we can rewrite the optimization in Eq. 3.21 into its *dual form* (while the original one is called the primal form) and just solve for n dual variables, which correspond to the Lagrange multipliers for the constraints. To derive the dual form, we introduce non-negative Lagrange multipliers $\boldsymbol{\alpha} = \{\alpha_i\}$ and $\boldsymbol{\nu} = \{\nu_i\}$ (for all $i \in \{1, \dots, n\}$) for each of the inequality constraints in 3.22 and 3.23, respectively. Recall that the rule is for constraints of the form constraint $_i \geq 0$, the constraint equations are multiplied by positive Lagrange multipliers and subtracted from the objective function, to form the Lagrangian. For equality constraints, the Lagrange multipliers are unconstrained. This gives the Lagrangian or dual form:

$$\mathcal{L}_{\text{dual}}(\boldsymbol{\theta}, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\nu}) = \mathcal{L}_{\text{primal}}(\boldsymbol{\theta}, \boldsymbol{\xi}) - \sum_{i=1}^n \alpha_i [y_i(\theta_0 + \boldsymbol{\theta}^T \mathbf{x}_i) - 1] - \sum_{i=1}^n \nu_i \xi_i$$

with Lagrange multipliers $\alpha_i \geq 0$ and $\nu_i \geq 0$ ($\forall i \in \{1, \dots, n\}$). The solution is given by the saddle point of the Lagrangian:

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\nu}} \min_{\boldsymbol{\theta}, \boldsymbol{\xi}} \mathcal{L}_{\text{dual}}(\boldsymbol{\theta}, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\nu}).$$

Applying the Karush–Kuhn–Tucker (KKT) conditions stated by (Fletcher, 1988), one obtains (i.e. by setting partial derivatives of the primal function with respect to variables to zero):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = 0 \rightarrow \boldsymbol{\theta} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = 0 \rightarrow \sum_{i=1}^n \alpha_i y_i = 0$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \rightarrow 0 \leq \alpha_i \leq C, \quad \forall i \in \{1, \dots, n\}$$

For simplicity, we only derive for the separable case in which the points can be perfectly separated into positive or negative (i.e. no need for the slack variables), while the non-separable

case can be done similarly. By substituting the terms $\theta = \sum_{i=1}^n \alpha_i y_i x_i$ and $\sum_{i=1}^n \alpha_i y_i = 0$ into the $\mathcal{L}_{\text{dual}}$, the dual problem can be simplified to a quadratic programming (QP) problem:

$$\max_{\boldsymbol{\alpha}} Q(\boldsymbol{\alpha}) \triangleq -\frac{1}{2} \sum_{i,j=1}^n y_i y_j \alpha_i \alpha_j x_i^T x_j + \sum_{i=1}^n \alpha_i$$

such that

$$\sum_{i=1}^n \alpha_i y_i = 0; \quad 0 \leq \alpha_i \leq C, \forall i \in \{1, \dots, n\}$$

that also leads to the so-called *box constraints* on $\boldsymbol{\alpha}$. Applying the kernel trick by replacing the inner product $x_i^T x_j$ by $k(x_i, x_j)$, we obtain the kernelized version of the QP:

$$\max_{\boldsymbol{\alpha}} Q(\boldsymbol{\alpha}) \triangleq -\frac{1}{2} \sum_{i,j=1}^n y_i y_j \alpha_i \alpha_j k(x_i, x_j) + \sum_{i=1}^n \alpha_i.$$

Standard solvers for QP takes $O(n^3)$ time. However, specialized algorithms, which avoid the use of generic QP solvers, have been developed for this problem including the *sequential minimal optimization* (SMO) algorithm proposed by (Platt, 1998) with time complexity $O(n^2)$. However, it can be still too slow for a large dataset. Therefore, it is popular to use linear SVMs (i.e. using the linear kernel), which only takes $O(n)$ time to train (Joachims, 2006) (Bottou et al., 2007).

The large margin principle

Here, we look at the SVM classification problem in a completely different perspective as we derive the primal function Eq. 3.21 by the *large margin principle*. Recall that our goal is to derive a discriminant parameterized function $f_\theta(x)$ which will be linear in the feature space implied by the choice of kernel, i.e. reproducing kernel Hilbert space (RKHS). We consider a point x and the decision boundary with normal vector θ (i.e. the set of points z such that $f_\theta(z) = 0$) in this induced space as $x = x_\perp + r \frac{\theta}{\|\theta\|}$, where r and x_\perp are the distance and the

orthogonal projection of x to the decision boundary. Thus,

$$f_\theta(x) = \theta_0 + \theta^T x = (\theta_0 + \theta^T x_\perp) + r \frac{\theta^T \theta}{\|\theta\|} = (\theta_0 + \theta^T x_\perp) + r \|\theta\|.$$

We also have $0 = f(x_\perp) = \theta_0 + \theta^T x_\perp$, so $f(x) = r \|\theta\|$ and $r = f(x)/\|\theta\|$. Intuitively,

- It is possible to have multiple hyperplanes that perfectly separate our training data, but among them, we want to select the one that maximizes the margin or the distance with the nearest point. For this reason, we say that SVM is an instance of a *large margin classifier*.
- Furthermore, we want that each point is on the correct side of the boundary, i.e. $f_\theta(x_i)y_i > 0$.

With this motivation, we have our new objective to maximize the margin as

$$\max_{\theta} \min_{i \in \{1, \dots, n\}} \frac{f_\theta(x_i)y_i}{\|\theta\|}.$$

It is important to note that:

- Scaling the parameters $\theta \rightarrow s\theta$ and $\theta_0 \rightarrow s\theta_0$ by a constant $s \in \mathbb{R}$ does not change the distance of any point to the boundary, because the factor s is canceled out when we divide by $\|\theta\|$.
- For the point that is closest to the decision boundary, we define the scale factor such that $f_\theta(x_i)y_i = 1$.
- Maximizing $1/\|\theta\|$ is the same as minimizing $\|\theta\|^2$.

Therefore, we derive our new objective

$$\min_{\theta} \frac{1}{2}\|\theta\|^2 \text{ s.t. } (\theta_0 + \theta^T x_i)y_i \geq 1, \forall i \in \{1, \dots, n\}.$$

The constraints mean that we force all points to be on the correct side of the decision boundary

with a margin of at least 1. If the data is not linearly separable, i.e. there is no feasible solution, then we must introduce *slack variables* $\xi_i \geq 0$ such that:

- If the point is on or inside the correct margin boundary: $\xi_i = 0$.
- Otherwise, $\xi_i = |y_i - f_\theta(x_i)|$. If the point lies inside the margin but still on the correct side of the decision boundary: $0 < \xi_i < 1$. If the point lies on the wrong side of the decision boundary then $\xi_i > 1$.

We replace the hard constraints that $f_\theta(x_i)y_i \geq 1$ with the *soft margin constraints* that $f_\theta(x_i)y_i \geq 1 - \xi_i$ to obtain the new objective:

$$\min_{\theta, \xi} C \sum_{i=1}^n \xi_i + \frac{1}{2} \|\theta\|^2 \quad \text{s.t. } (\theta_0 + \theta^T x_i) y_i \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad (\forall i \in \{1, \dots, n\})$$

that is fundamentally the same as in Eq. 3.21 but is derived in an alternative way. The first term $\sum_{i=1}^n \xi_i$ is an upper-bound on the number of misclassified points, since $\xi_i > 1$ if x_i is misclassified. It is popular to define the regularization parameter $C = 1/(\nu n)$, where ν controls the fraction of misclassified points during training. Usually ν is a hyperparameter to search for via cross-validation. This model is called a ν -SVM classifier.

SVMs for multi-class classification

Upgrading SVM from binary classification to the multi-class case is not so trivial. There are two common approaches:

1. **One-vs-all:** Suppose that we have L different discrete labels/classes. We train L binary classifiers, $f_{\theta_\ell}(x)$ such that each classifier has its own parameters θ_ℓ , and the data from class ℓ is treated as positive while the rest is treated as negative. Then for the prediction, we select $\hat{y}(x) = \arg \max_\ell f_{\theta_\ell}(x)$. Limitations of this approach include the inputs are ambiguously relabeled, there is no guarantee that different f_{θ_ℓ} functions have comparable magnitudes, and it is likely that each binary problem will suffer from the *class imbalance* issue (i.e. the number of negative examples is much more than the

number of positive ones). The time complexities for this approach are $O(Ln^2)$ for training and $O(Lm)$ for testing where m is the number of support vectors.

2. **One-vs-one:** We train $L(L - 1)/2$ classifiers to discriminate all pairs of labels (ℓ, ℓ') , then we classify a point into the class which has the highest number of votes. One limitation of this approach is also ambiguities. The time complexities are $O(L^2n^2)$ for training and $O(L^2m)$ for testing where m is the number of support vectors. However, if we suppose that the labels are uniformly distributed in our dataset, then each label has roughly n/L points, so the training complexity is about $O(n^2)$ only.

Interested readers in support vector machines are recommended to read (Burges, 1998) and (Murphy, 2012b) for further reference.

3.3.4 Kernel principal component analysis

Without graph neural networks, producing a meaningful low-dimensional Euclidean representation for combinatorics structures like graphs is indeed a challenge. However, with graph kernels and the kernel trick that defines our feature vector in terms of kernels, $\phi(x) = [k(x, x_1), \dots, k(x, x_n)]^T$, we have statistical ways to achieve vectorized graph representation. The technique we present in this section is *kernel principal component analysis* (Kernel PCA) proposed by (Schölkopf et al., 1999) as a kernelized extension of the conventional PCA that computes a low-dimensional linear embedding of a given data. The original PCA can be understood as kernel PCA with linear or inner-product kernel.

First, we will discuss a useful trick for linear PCA for the case when $d > n$ (i.e. the number of features is larger than the number of data points). This trick can help us to understand the kernel trick and how to produce a nonlinear embedding later. PCA requires finding the eigenvectors of the sample covariance matrix $\mathbf{S} = \frac{1}{n}x_i x_i^T = \frac{1}{n}\mathbf{X}^T \mathbf{X}$. But we can also compute PCA by finding the eigenvectors \mathbf{U} (i.e. we store all eigenvectors $\{u_j\}$ column-wise

in this matrix) and the corresponding eigenvalues Λ (i.e. we store all eigenvalues $\{\lambda_j\}$ in the diagonal of this matrix) of the inner-product gram matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^T$. By definition, $\mathbf{K}\mathbf{U} = \mathbf{U}\Lambda$. Multiplying both sides with \mathbf{X}^T , we get

$$\mathbf{X}^T \mathbf{K} \mathbf{U} = \mathbf{X}^T \mathbf{U} \Lambda \Leftrightarrow (\mathbf{X}^T \mathbf{X}) \mathbf{X}^T \mathbf{U} = (\mathbf{X}^T \mathbf{U}) \Lambda \Leftrightarrow \mathbf{S} (\mathbf{X}^T \mathbf{U}) = (\mathbf{X}^T \mathbf{U}) \Lambda,$$

that means \mathbf{S} have eigenvectors $\mathbf{V} = \mathbf{X}^T \mathbf{U}$ with eigenvalues Λ . Because the squared norm of each individual eigenvector v_j of \mathbf{S} is $\|v_j\|^2 = u_j^T \mathbf{X} \mathbf{X}^T u_j = \lambda_j u_j^T u_j = \lambda_j$, the normalized eigenvectors are given by

$$\mathbf{V}_{\text{pca}} = \mathbf{X}^T \mathbf{U} \Lambda^{-1/2}.$$

In a similar way, now we can extend linear PCA to kernel PCA as follows. Mercer's theorem (Mercer, 1909) states that the use of a kernel $k(\cdot, \cdot)$ implies some underlying feature space (e.g., feature map $\phi(\cdot)$ into a possibly infinite dimensional Hilbert space). Implicitly, we replace x_i with $\phi(x_i)$. We denote the corresponding design matrix as Φ and the covariance matrix in feature space as $\mathbf{S}_\phi = \frac{1}{n} \sum_{i=1}^n \phi(x_i) \phi(x_i)^T$. The eigenvectors are given by

$$\mathbf{V}_{\text{kPCA}} = \Phi^T \mathbf{U} \Lambda^{-1/2},$$

but we cannot actually compute it because the feature map $\phi(x_i)$ can be in an infinite dimensional space. Fortunately, we can compute the projection of a test vector x_* onto the feature space as follows:

$$\phi(x_*)^T \mathbf{V}_{\text{kPCA}} = \phi(x_*)^T \Phi^T \mathbf{U} \Lambda^{-1/2} = \mathbf{k}_*^T \mathbf{U} \Lambda^{-1/2},$$

where

$$\mathbf{k}_* = [\langle \phi(x_*), \phi(x_1) \rangle, \dots, \langle \phi(x_*), \phi(x_n) \rangle]^T = [k(x_*, x_1), \dots, k(x_*, x_n)]^T.$$

So far, we assumed that the projected data has zero mean, but we cannot simply subtract off the mean in feature space as we usually do in Euclidean case. Again, we employ one more kernel trick. We define the centered feature vector as

$$\tilde{\phi}(x_i) = \phi(x_i) - \frac{1}{n} \sum_{j=1}^n \phi(x_j).$$

The kernel gram matrix of these centered vectors $\tilde{\mathbf{K}}$ has each element $\tilde{K}_{ij} = \tilde{\phi}(x_i)^T \tilde{\phi}(x_j)$ is given by

$$\tilde{K}_{ij} = \phi(x_i)^T \phi(x_j) - \frac{1}{n} \sum_{a=1}^n \phi(x_i)^T \phi(x_a) - \frac{1}{n} \sum_{a=1}^n \phi(x_j)^T \phi(x_a) + \frac{1}{n^2} \sum_{a=1}^n \sum_{b=1}^n \phi(x_a)^T \phi(x_b),$$

or equivalently,

$$\tilde{K}_{ij} = k(x_i, x_j) - \frac{1}{n} \sum_{a=1}^n k(x_i, x_a) - \frac{1}{n} \sum_{a=1}^n k(x_j, x_a) + \frac{1}{n^2} \sum_{a=1}^n \sum_{b=1}^n k(x_a, x_b).$$

We can express $\tilde{\mathbf{K}}$ in matrix notation as

$$\tilde{\mathbf{K}} = \mathbf{H} \mathbf{K} \mathbf{H},$$

where $\mathbf{H} \triangleq \mathbf{I} - \frac{1}{n} \mathbf{1} \mathbf{1}^T$ is the *centering matrix* with $\mathbf{1} = [1, \dots, 1]^T$ is a vector of n 1s. While linear PCA is limited to using $r \leq d$ components, in kernel PCA, we can use up to n components because the size of Φ is $n \times d'$ where d' is the potentially infinite dimensionality of embedded features vectors in a Hilbert space.

Once we obtain the vectorized representation of graphs via kernel PCA on a graph kernel gram matrix, we can do unsupervised clustering or visualization on this graph dataset. Furthermore, it is possible to apply linear regression or logistic regression on these vectorized representations for the tasks of graph properties regression or graph classification, respec-

tively. Some simple extensions of the traditional methods such as kernelized nearest neighbor classification and kernelized K-medoids clustering have been proposed in the field of statistical learning. In general, Gaussian Processes and Support Vector Machines might be able to give a better result in the supervised setting. Some useful references include (Hastie et al., 2001), (Bishop, 2006), and (Murphy, 2012b). In the next section, we will discuss the learning and neural networks approach for graph representation learning that recently outperformed previously proposed statistical machine learning methods on graphs.

3.4 Message Passing Neural Networks and its variants

Recently, with the advent of deep learning and much larger datasets, a sequence of neural network based approaches have appeared to address the problem of graph representation learning, starting with (Scarselli et al., 2009). In contrast to the kernels framework, neural networks effectively integrate the classification or regression problem at hand with learning the graph representation itself, in a single, end-to-end system. In the last few years, there has been a veritable explosion in research activity in this area. Some of the proposed graph learning architectures (Duvenaud et al., 2015; Kearnes et al., 2016; Niepert et al., 2016) directly seek inspiration from the type of classical CNNs that are used for image recognition (LeCun et al., 1998; Krizhevsky et al., 2012). These methods involve first fixing a vertex ordering, then moving a filter across vertices while doing some computation as a function of the local neighborhood to generate a representation. This process is then repeated multiple times like in classical CNNs to build a deep graph representation. Other notable works on graph neural networks include (Li et al., 2016a; Schütt et al., 2017; Battaglia et al., 2016; Kipf and Welling, 2017). Very recently, (Gilmer et al., 2017) showed that many of these approaches can be seen to be specific instances of a general message passing formalism, and coined the term *message passing neural networks* (MPNNs) to refer to them collectively. In this section, we introduce the well-known MPNNs and its several variants.

3.4.1 Label propagation algorithm

Label propagation algorithm, or in general the Message Passing framework, has been widely applied to various network problems ranging from PageRank for Google search engine to learning representations for molecules with graph neural networks. The core idea can be simply explained in words as follows. Given an input graph / network $G = (V, E)$. Initially, each vertex v of the graph is associated with a feature representation l_v (label) or f_v^0 (e.g., scalar in PageRank, vector in majority of graph neural networks, or a higher-order tensor in Covariant Compositional Networks). This feature representation can also be called as a *message*. Iteratively, at iteration ℓ , each vertex collects / aggregates all messages of the previous iteration $\{f_{v_1}^{\ell-1}, \dots, f_{v_k}^{\ell-1}\}$ from other vertices in its neighborhood $\mathcal{N}(v) = \{v_1, \dots, v_k\}$, and then produces a new message f_v^ℓ via some *hashing function* $\Phi(\cdot)$. The graph representation $\phi(G)$ is obtained by aggregating all messages in the last iteration of every vertex. The generic algorithm is described in pseudocode 8.

Algorithm 8 Label Propagation Algorithm

```

1: for  $v \in V$  do
2:    $f_v^0 \leftarrow l_v$ 
3: end for
4: for  $\ell = 1 \rightarrow L$  do
5:   for  $v \in V$  do
6:      $f_v^\ell \leftarrow \Phi(f_{v_1}^{\ell-1}, \dots, f_{v_k}^{\ell-1})$  where  $\mathcal{N}(v) = \{v_1, \dots, v_k\}$ 
7:   end for
8: end for
9:  $\phi(G) \leftarrow \Phi(f_1^L, \dots, f_{|V|}^L)$ 
10: Use  $\phi(G)$  for downstream regression / classification tasks.

```

3.4.2 Generic graph neural network

Graph neural networks can be built based on the Message Passing framework in which the hashing function $\Phi(\cdot)$ at iteration ℓ has n_ℓ learnable parameters $\{W_1^\ell, \dots, W_{n_\ell}^\ell\}$. The gradients of the loss function with respect to these learnable parameters can be computed by Back-

Propagation algorithm, similarly to Recurrent Neural Networks. The learnable parameters will then be optimized by Stochastic Gradient Descent (SGD) or its variants. It is required that $\Phi(\cdot)$ must be differentiable. The generic algorithm of a graph neural network is described in pseudocode 9. In later sections of this chapter, we will go through a brief summary for the field of graph neural networks with state-of-the-art algorithms including Neural Graph Fingerprint (see section 3.4.3), Convolutional Neural Networks for graphs (see section 3.4.4), Gated Graph Neural Networks (see section 3.4.5), Weisfeiler-Lehman Networks (see section 3.4.6), Message Passing Neural Networks (see section 3.4.7), Interaction Networks (see section 3.4.8), Molecular Graph Convolutions (see section 3.4.9), Deep Tensor Neural Networks (see section 3.4.10).

Algorithm 9 Generic Graph Neural Network

- 1: Initialize learnable parameters $\{W_1^\ell, \dots, W_{n_\ell}^\ell\}$ for each layer ℓ .
 - 2: Initialize learnable parameters $\{W_1, \dots, W_n\}$ for learning the graph representation.
 - 3: **for** $v \in V$ **do**
 - 4: $f_v^0 \leftarrow l_v$
 - 5: **end for**
 - 6: **for** $\ell = 1 \rightarrow L$ **do**
 - 7: **for** $v \in V$ **do**
 - 8: $f_v^\ell \leftarrow \Phi(f_{v_1}^{\ell-1}, \dots, f_{v_k}^{\ell-1}; \{W_1^\ell, \dots, W_{n_\ell}^\ell\})$ where $\mathcal{N}(v) = \{v_1, \dots, v_k\}$
 - 9: **end for**
 - 10: **end for**
 - 11: $\phi(G) \leftarrow \Phi(f_1^L, \dots, f_{|V|}^L; \{W_1, \dots, W_n\})$
 - 12: Use $\phi(G)$ for downstream regression / classification tasks.
-

3.4.3 Neural graph fingerprint

Given an input graph $G = (V, E, A)$, where V is the set of vertices, E is the set of edges and matrix $A \in \{0, 1\}^{|V| \times |V|}$ is the corresponding adjacency matrix. The goal is to learn an unknown class of functions parameterized by $\{W_1, \dots, W_L, u\}$ in the following scheme:

1. The inputs are vectors $f(v) \in \mathbb{R}^d$ for each vertex $v \in V$. We call the vector embedding f the multi-dimensional vertex label function.

2. We assume some learnable weight matrix $W_\ell \in \mathbb{R}^{d \times d}$ associating with level ℓ -th of the neural network. For L levels, we update the vector stored at vertex v using W_ℓ .
3. Finally, we assume some learnable weight vector $u \in \mathbb{R}^d$. We add up the iterated vertex labels and dot product the result with u . This can be considered as a linear regression on top of the graph neural network.

More formally, we define the L -iteration label propagation algorithm on graph G . Let $h_\ell(v) \in \mathbb{R}^d$ be the vertex embedding of vertex v at iteration $\ell \in \{0, \dots, L\}$. At $\ell = 0$, we initialize $h_0(v) = f(v)$. At $\ell \in \{1, \dots, L\}$, we update $h_{\ell-1}$ to h_ℓ at a vertex v using the values on v 's neighbors:

$$h_\ell(v) = h_{\ell-1}(v) + \frac{1}{|\mathcal{N}(v)|} \sum_{w \in \mathcal{N}(v)} h_{\ell-1}(w) \quad (3.24)$$

where $\mathcal{N}(v) = \{w \in V | (v, w) \in E\}$ denotes the set of adjacent vertices to v . We can write the label propagation algorithm in a matrix form. Let $H_\ell \in \mathbb{R}^{|V| \times d}$ denote the vertex embedding matrix in which the v -th row of H_ℓ is the embedding of vertex v at iteration ℓ . Equation 3.24 is equivalent to:

$$H_\ell = (I_{|V|} + D^{-1} \cdot A) \cdot H_{\ell-1} \quad (3.25)$$

where $I_{|V|}$ is the identity matrix of size $|V| \times |V|$ and D is the diagonal matrix with entries equal to the vertex degrees. Note that's is also common to define another label propagation algorithm via the normalized graph Laplacian (Kipf and Welling, 2017):

$$H_\ell = (I_{|V|} - D^{-1/2} A D^{-1/2}) \cdot H_{\ell-1} \quad (3.26)$$

From the label propagation algorithms, we build the simplest form of graph neural networks (Kearnes et al., 2016; Duvenaud et al., 2015; Kipf and Welling, 2017). Suppose that iteration ℓ is associated with a learnable matrix $W_\ell \in \mathbb{R}^{d \times d}$ and a component-wise nonlinearity function σ ; in our case σ is the sigmoid function. We imagine that each iteration now becomes a layer

of the graph neural network. We assume that each graph G has input labels f and a learning target $\mathcal{T}_G \in \mathbb{R}$. The forward pass of the graph neural network (GNN) is described by algorithm 10.

Algorithm 10 Forward pass of GNN

- 1: **Input:** Given an undirected graph $G = (V, E, A)$ where V , E and A are the set of vertices, the set of edges and the adjacency matrix, respectively. The number of layers is $L \in \mathbb{N}$.
 - 2: **Output:** Construct the corresponding neural network.
 - 3: Initialize $W_0, W_1, \dots, W_L \in \mathbb{R}^{d \times d}$
 - 4: Layer 0: $\mathcal{L}_0 = \sigma(H_0 \cdot W_0)$
 - 5: Layer $\ell \in \{1, \dots, L\}$: $\mathcal{L}_\ell = \sigma(H_\ell \cdot W_\ell)$
 - 6: Compute the graph feature: $f_G = \sum_{v \in V} \mathcal{L}_L(v) \in \mathbb{R}^d$
 - 7: Linear regression on layer $L + 1$
 - 8: Minimize $\|\langle u, f_G \rangle - \mathcal{T}_G\|_2^2$ where $u \in \mathbb{R}^d$ is learnable
-

Learnable matrices W_ℓ and learnable vector u are optimized by the Back-Propagation algorithm as done when training a conventional multi-layer feed-forward neural network.

To empower Neural Graph Fingerprint, we can also introduce quadratic and cubic aggregation rules that can be considered a special simplified form of tensor contractions. In detail, the linear aggregation rule can be defined as summation of feature vectors in a neighborhood $\mathcal{N}(v)$ of vertex v at level $\ell - 1$ to get a permutation invariant representation of vertex v at level ℓ :

$$\phi_\ell^{linear}(v) = \sum_{w \in \mathcal{N}(v)} h_{\ell-1}(w)$$

where $\phi_\ell^{linear}(v) \in \mathbb{R}^d$ and $h_{\ell-1}(w) \in \mathbb{R}^d$ are still in zeroth order representation such that each channel of d channels is represented by a single scalar. Extending this we get the quadratic aggregation rule for $\phi_\ell^{quadratic}(v)$:

$$\phi_\ell^{quadratic}(v) = diag \left(\sum_{u \in \mathcal{N}(v)} \sum_{w \in \mathcal{N}(v)} h_{\ell-1}(u) h_{\ell-1}(w)^T \right)$$

where $h_{\ell-1}(u)h_{\ell-1}(w)^T \in \mathbb{R}^{d \times d}$ is the outer-product of level $(\ell - 1)$ -th representation of vertex u and w in the neighborhood $\mathcal{N}(v)$. Again $\phi_\ell^{quadratic}(v) \in \mathbb{R}^d$ is still in zeroth-order. Finally, we extend to the cubic aggregation rule for $\phi_\ell^{cubic}(v)$:

$$\phi_\ell^{cubic}(v) = \text{diag}\left(\sum_{u,w,t \in \mathcal{N}(v)} h_{\ell-1}(u) \otimes h_{\ell-1}(w) \otimes h_{\ell-1}(t)\right)$$

where $h_{\ell-1}(u) \otimes h_{\ell-1}(w) \otimes h_{\ell-1}(t) \in \mathbb{R}^{d \times d \times d}$ is the tensor product of 3 rank-1 vectors, and we obtain zeroth-order $\phi_\ell^{cubic}(v) \in \mathbb{R}^d$ by taking the diagonal of the 3rd order result tensor.

Moreover, it is not a natural idea to limit the neighborhood $\mathcal{N}(v)$ to only the set of adjacent vertices of v . Another way to extend $\mathcal{N}(v)$ is to use different neighborhoods at different levels / layers of the network, for example:

- At level $\ell = 0$: $\mathcal{N}_0(v) = \{v\}$

- At level $\ell > 0$:

$$\mathcal{N}_\ell(v) = \mathcal{N}_{\ell-1}(v) \cup \bigcup_{w \in B(v,1)} \mathcal{N}_{\ell-1}(w)$$

where $B(v,1)$ denotes the set of vertices are at the distance 1 from the center v .

From equation 3.25, we extend the basic GNN as follows. Let \bar{A} be the normalized adjacency matrix (or probability transition matrix) in which \bar{A}_{ij} corresponds to the transition probability from vertex i to vertex j (via only 1 edge). Similarly, $[\bar{A}]_{ij}^k$ corresponds to the probability of a random walk starting at vertex i and ending at vertex j after k edges. Using \bar{A}^k means for each vertex, we consider its neighborhood of distance k by a walk. We have the extension of 3.25:

$$H_\ell = \left(\sum_{k=0}^n \bar{A}^k \right) \cdot H_{\ell-1} \quad (3.27)$$

where $\bar{A}^0 \equiv I_{|V|}$ and n is the maximum distance for the neighborhood.

3.4.4 Learning convolutional neural networks for graphs

The idea of Learning Convolutional Neural Networks for Graphs (LCNN) from (Niepert et al., 2016) can be summarized as *flattening* a graph into a fixed-size sequence. Suppose that the maximum number of vertices over the whole dataset is N . Consider an input graph $G = (V, E)$. If $|V| < N$ then we add $N - |V|$ *dummy vertices* into V such that every graph in the dataset has the same number of vertices. For each vertex $v \in V$, LCNN fixes the size of its neighborhood $\Omega(v)$ as K . In the case $|\Omega(v)| < K$, again we add $K - |\Omega(v)|$ dummy vertices into $\Omega(v)$ to ensure that every neighborhood of every vertex has exactly the same number of vertices. Let $d : V \times V \rightarrow \{0, \dots, |V| - 1\}$ denote the shortest-path distance between any pair of vertices in G . Let $\sigma : V \rightarrow \mathbb{R}$ denote the sub-optimal hashing function obtained from Weisfeiler-Lehman graph isomorphism test. Based on σ , we can obtain a sub-optimal ranking of vertices. The neighborhood $\Omega(v)$ of vertex v is constructed by algorithm 11. We

Algorithm 11 Construct Neighborhood of $v \in V$

```

1: Input: Given an undirected graph  $G = (V, E, A)$  and a vertex  $v \in V$ .
2: Output: Construct the receptive field  $\Omega(v)$ .
3:  $\Omega(v) \leftarrow \emptyset$ 
4: for  $l \in 0, \dots, |V| - 1$  do
5:   for  $w \in V$  do
6:     if  $d(v, w) = l$  then
7:        $\Omega(v) \leftarrow \Omega(v) \cup \{w\}$ 
8:     end if
9:   end for
10:  if  $|\Omega(v)| \geq K$  then
11:    break
12:  end if
13: end for
14: if  $|\Omega(v)| < K$  then
15:   Add  $K - |\Omega(v)|$  dummy vertices into  $\Omega(v)$ 
16: end if
17: Suppose  $\Omega(v) = \{v_1, \dots, v_K\}$ 
18: Sort  $\Omega(v) \leftarrow \{v_{i_1}, \dots, v_{i_K}\}$  such that  $\sigma(v_{i_t}) < \sigma(v_{i_{t+1}})$ 
19: Return:  $\Omega(v)$ 

```

also have algorithm 12 to flatten the input graph G as follows into a sequence of $N \times K$

vertices.

Algorithm 12 Flattening the graph

- 1: **Input:** Given an undirected graph $G = (V, E)$.
 - 2: **Output:** Sequence S of $N \times K$ vertices.
 - 3: Suppose that $V = \{v_1, \dots, v_{|V|}\}$
 - 4: Sort $\bar{V} \leftarrow \{v_{i_1}, \dots, v_{i_{|V|}}\}$ such that $\sigma(v_{i_t}) < \sigma(v_{i_{t+1}})$
 - 5: Initialize sequence $S \leftarrow \emptyset$
 - 6: **for** $v \in \bar{V}$ **do**
 - 7: Add $\Omega(v)$ at the end of S
 - 8: **end for**
 - 9: Return S
-

Suppose that each vertex is associated with a fixed-size input feature vector of L channels.

By the graph flattening algorithm 12, we can produce a feature matrix of size $L \times (NK)$.

We can apply the standard convolutional operation as 1D Convolutional Neural Network on the columns of this matrix. On top of LCNN is a fully-connected layer for regression tasks or classification tasks.

3.4.5 Gated graph neural networks

Long Short-Term Memory (LSTM), firstly proposed by (Hochreiter and Schmidhuber, 1997), is a special kind of Recurrent Neural Network that was designed for learning sequential and time-series data. LSTM is widely applied into many current state-of-the-art Deep Learning models in various aspects of Machine Learning including Natural Language Processing, Speech Recognition and Computer Vision. Gated Recurrent Unit (GRU) was introduced by (Kyunghyun et al., 2014) in their EMNLP 2014 paper in the context of sequential modeling. GRU can be understood as a simplification of LSTM.

With the spirit of Language Modeling, throughout the neural network, from level 0 to level

L , all representations of a vertex v can be represented as a sequence:

$$f_v^{(0)} \rightarrow f_v^{(1)} \rightarrow \dots \rightarrow f_v^{(L)}$$

in which $f_v^{(\ell)}$ is more global than $f_v^{(\ell-1)}$, and $f_v^{(\ell-1)}$ is more local than $f_v^{(\ell)}$. One can think of the sequence of representations as a sentence of words as in Natural Language Processing. We can embed GRU / LSTM at each level of our network in the sense that GRU / LSTM at level ℓ will learn to choose whether to select $f_v^{(\ell)}$ as the final representation or reuse one of the previous level representations $\{f_v^{(0)}, \dots, f_v^{(\ell-1)}\}$. This idea is inherited from Gated Graph Neural Networks (GGNN) of (Li et al., 2016a) in ICLR 2016. The algorithm (propagation model) of GGNN with GRU is described in pseudocode 13.

3.4.6 Weisfeiler-Lehman network

Weisfeiler-Lehman Network (WLN) inspired by the Weisfeiler-Lehman isomorphism test for labeled graphs is proposed by (Jin et al., 2017) in their NIPS 2017 paper in the context of molecular graphs. The architecture is designed to embed the computations inherent in WL isomorphism testing to generate learned isomorphism-invariant representations for atoms. Let $c_v^{(L)}$ be the final label of atom a_v where L is the number of levels/layers in WLN. The molecular graph $G = (V, E)$ is represented as a set $\{(c_u^{(L)}, b_{uv}, c_v^{(L)}) | (u, v) \in E\}$, where b_{uv} is the bond type between u and v . Let r be the analogous continuous relabeling function. Then a node $v \in G$ with neighbor nodes $\mathcal{N}(v)$, node features f_v , and edge features f_{uv} is relabeled according to:

$$r(v) = \tau\left(U_1 f_v + U_2 \sum_{u \in \mathcal{N}(v)} \tau(V[f_u, f_{uv}])\right) \quad (3.28)$$

Algorithm 13 Gated Graph Neural Network

- 1: **Input:** Given an undirected graph $G = (V, E, A)$ where V , E and A are the set of vertices, the set of edges and the adjacency matrix, respectively. The number of layers is $L \in \mathbb{N}$. Each vertex $v \in V$ is associated with an input feature vector x_v .
 - 2: **Output:** Construct the corresponding neural network.
 - 3: **Notions:**
 - $a_v^{(\ell)}$: Aggregated message of vertex v at level ℓ from its neighborhood vertices $\mathcal{N}(v)$.
 - $z_v^{(\ell)}, r_v^{(\ell)}$: Forget and update gates of GRU.
 - σ : Sigmoid function.
 - $\overline{f_v^{(\ell)}}$: Proposed output of vertex v at level ℓ .
 - \odot : Component-wise multiplication.
 - $f_v^{(\ell)}$: Final output (representation) of vertex v at level ℓ .
 - f_G : Graph representation.
 - $i(\cdot), j(\cdot)$: Multi-layer perceptron.
 - 4: Initialize $f_v^{(0)}$ from x_v for all vertex v .
 - 5: Initialize learnable weight matrices W^z, U^z, W^r, U^r, W and U .
 - 6: **for** $\ell = 1 \rightarrow L$ **do**
 - 7: **for** $v \in V$ **do**
 - 8: $a_v^{(\ell)} = \sum_{v' \in \mathcal{N}(v)} f_{v'}^{(\ell-1)}$
 - 9: $z_v^{(\ell)} = \sigma(W^z a_v^{(\ell)} + U^z f_v^{(\ell-1)})$
 - 10: $r_v^{(\ell)} = \sigma(W^r a_v^{(\ell)} + U^r f_v^{(\ell-1)})$
 - 11: $\overline{f_v^{(\ell)}} = \tanh(W a_v^{(\ell)} + U(r_v^{(\ell)} \odot f_v^{(\ell-1)}))$
 - 12: $f_v^{(\ell)} = (1 - z_v^{(\ell)}) \odot f_v^{(\ell-1)} + z_v^{(\ell)} \odot \overline{f_v^{(\ell)}}$
 - 13: **end for**
 - 14: **end for**
 - 15: $f_G = \tanh\left(\sum_{v \in V} \sigma(i(f_v^{(L)}, x_v)) \odot \tanh(j(f_v^{(L)}, x_v))\right)$
 - 16: Use the graph feature f_G for downstream tasks.
-

where $\tau(\cdot)$ could be any component-wise non-linearity function. We apply this relabeling operation iteratively to obtain context-dependent atom vectors for $1 \leq \ell \leq L$:

$$h_v^{(\ell)} = \tau\left(U_1 h_v^{(\ell-1)} + U_2 \sum_{u \in \mathcal{N}(v)} \tau(V[h_u^{(\ell-1)}, f_{uv}])\right) \quad (3.29)$$

where $h_v^{(0)} = f_v$ and U_1, U_2, V are learnable weight matrices shared across layers. The final atom representations arise from mimicking the set comparison function in the WL isomorphism test, yielding:

$$c_v = \sum_{u \in \mathcal{N}(v)} W^{(0)} h_u^{(L)} \odot W^{(1)} f_{uv} \odot W^{(2)} h_v^{(L)} \quad (3.30)$$

The set comparison here is realized by matching each rank-1 edge tensor $h_u^{(L)} \otimes f_{uv} \otimes h_v^{(L)}$ to a set of reference edges also cast as rank-1 tensors $W^{(0)}[k] \otimes W^{(1)}[k] \otimes W^{(2)}[k]$, where $W[k]$ is the k -th row of matrix W . In other words, equation 3.30 could be written as:

$$c_v[k] = \sum_{u \in \mathcal{N}(v)} \langle W^{(0)}[k] \otimes W^{(1)}[k] \otimes W^{(2)}[k], h_u^{(L)} \otimes f_{uv} \otimes h_v^{(L)} \rangle \quad (3.31)$$

The resulting c_v is a vector representation that captures the local chemical environment of the atom (through relabeling) and involves a comparison against a learned set of reference environments. The representation of the whole graph G is simply the sum over all the atom representations, and will be used in downstream regression/classification tasks:

$$c_G = \sum_{v \in V} c_v \quad (3.32)$$

3.4.7 Message Passing Neural Networks

(Gilmer et al., 2017) reintroduced the existing Message Passing framework in the context of neural networks with an application in quantum chemistry as estimating the solution of Density Functional Theory. Message Passing Neural Networks (MPNNs) operate on undirected graphs G with node features x_v and edge features e_{vw} . The forward pass is divided into two phases: message passing phase and readout phase. In the message passing phase, the iterative algorithm executes for L time steps (or layers) and is defined in terms of message functions M_ℓ and vertex update functions U_ℓ . At layer ℓ , each vertex v is associated with a hidden state (vertex representation) h_v^ℓ and is updated based on the message m_v^ℓ according to:

$$m_v^{\ell+1} = \sum_{w \in \mathcal{N}(v)} M_\ell(h_v^\ell, h_w^\ell, e_{vw}) \quad (3.33)$$

$$h_v^{\ell+1} = U_\ell(h_v^\ell, m_v^{\ell+1}) \quad (3.34)$$

where $\mathcal{N}(v)$ denotes the neighbors of v in graph G . The readout phase computes a feature vector $\phi(G)$ for the whole graph using some readout function R according to:

$$\phi(G) = R(\{h_v^L | v \in G\}) \quad (3.35)$$

The message functions M_ℓ , vertex update functions U_ℓ , and readout function R are all learned differentiable functions with learnable parameters (e.g., Multi-layer Perceptron or MLP). R operates on the set of vertex representations and is required to be invariant to vertex permutations in order for the MPNN to be permutation-invariant (invariant to graph isomorphism). Many models in the literature (not all) can be projected into the MPNN framework by specifying the message functions M_ℓ , vertex update functions U_ℓ , and readout function R . In addition, MPNN can be easily extended to learn edge representations by introducing hidden states $h_{e_{vw}}^\ell$ for all edges in the graph and updating them analogously to

equations 3.33 and 3.34.

3.4.8 Interaction networks

Interaction Networks (IN) proposed by (Battaglia et al., 2016) considers the graph learning problem in which each vertex and the whole graph are associated with learning targets. In the language of MPNN framework, the message function $M(h_v, h_w, e_{vw})$ and update function $U(h_v, x_v, m_v)$ of IN are MLPs taking the inputs as a vector concatenation. The graph level output is $R = f(\sum_{v \in G} h_v^L)$ where f is an MLP which takes the sum of the final vertex representation h_v^L . In the original work, the authors only considered the model with $L = 1$.

3.4.9 Molecular graph convolutions

Molecular Graph Convolutions introduced by (Kearnes et al., 2016) is based on the MPNN framework in which edge representations are updated during the message passing phase. The vertex update function is:

$$U_\ell(h_v^\ell, m_v^{\ell+1}) = \alpha(W_1(\alpha(W_0 h_v^\ell), m_v^{\ell+1})) \quad (3.36)$$

where $(.,.)$ denotes the concatenation operation, α is the RELU activation. The edge update function is:

$$e_{vw}^{\ell+1} = U'_\ell(e_{vw}^\ell, h_v^\ell, h_w^\ell) = \alpha(W_4(\alpha(W_2 e_{vw}^\ell), \alpha(W_3(h_v^\ell, h_w^\ell)))) \quad (3.37)$$

In both equations 3.36 and 3.37, W_i are learnable weight matrices.

3.4.10 Deep tensor neural networks

Deep Tensor Neural Networks (DTNN) proposed by (Schütt et al., 2017) focuses on physical systems of particles that can be interpreted as complete graphs. DTNN computes the

message from atom w to atom v by:

$$M_\ell = \tanh \left(W^{fc} ((W^{cf} h_w^\ell + b_1) \odot (W^{df} e_{vw} + b_2)) \right) \quad (3.38)$$

where W^{fc} , W^{cf} , and W^{df} are matrices and b_1 , b_2 are bias vectors. The vertex update function is:

$$h_v^{\ell+1} = U_\ell(h_v^\ell, m_v^{\ell+1}) = h_v^\ell + m_v^{\ell+1} \quad (3.39)$$

The readout function is defined as:

$$\phi(G) = \sum_{v \in G} NN(h_v^L) \quad (3.40)$$

where $NN(h_v^L)$ is a single hidden layer neural network (fully-connected) taking input as the hidden state of atom v from the last layer.

3.5 Learning graphs on the spectral domain

Family of models learning graphs on the spectral domain based on Graph Fourier Transform have been proposed by (Defferrard et al., 2016) and (Bruna et al., 2014) that can also be seen as instances of graph neural networks. These models generalize the notion of convolutions on a general graph G with N vertices. Given an adjacency matrix $A \in \mathbb{R}^{N \times N}$. The graph Laplacian is defined as:

$$L = I - D^{-1/2} A D^{-1/2} \quad (3.41)$$

where D is the diagonal degree matrix. Let V denote the eigenvectors of L , ordered by eigenvalue. Let σ be a real-valued nonlinearity. The fundamental operation in this family of models is the Graph Fourier Transformation. We define an operation which transforms an input vector x of size $N \times d_1$ to an output vector y of size $N \times d_2$ (we transform each vertex

representation of size d_1 into size d_2):

$$y_j = \sigma \left(\sum_{i=1}^{d_1} V F_{i,j} V^T x_i \right) \quad (j \in \{1, \dots, d_2\}) \quad (3.42)$$

The matrices $F_{i,j}$ are all diagonal $N \times N$ matrices and contain all of the learnable parameters in the layer. Define the rank 4 tensor \hat{L} of dimension $N \times N \times d_1 \times d_2$ where $\hat{L}_{v,w,i,j} = (V F_{i,j} V^T)_{v,w}$ in which v, w are the indices corresponding to vertices. Let $\hat{L}_{v,w}$ denote the $d_1 \times d_2$ dimensional matrix where $(\hat{L}_{v,w})_{i,j} = \hat{L}_{v,w,i,j}$. Equation 3.42 can be written as:

$$y_{v,j} = \sigma \left(\sum_{i=1, w=1}^{d_1, N} \hat{L}_{v,w,i,j} x_{w,i} \right) \quad (3.43)$$

or in short:

$$y_{v,:} = \sigma \left(\sum_{w=1}^N \hat{L}_{v,w} x_w \right) \quad (3.44)$$

We relabel y_v as $h_v^{\ell+1}$ and x_w as h_w^ℓ . In the language of the MPNN framework, the message update function can be written as:

$$m_v^{\ell+1} = M(h_v^\ell, h_w^\ell) = \hat{L}_{v,w} h_w^\ell \quad (3.45)$$

and the vertex update function can be written as:

$$h_v^{\ell+1} = U(h_v^\ell, m_v^{\ell+1}) = \sigma(m_v^{\ell+1}) \quad (3.46)$$

3.6 Graph-based Semi-Supervised Learning

Graph-based Semi-Supervised Learning (SSL) was proposed by (Ravi and Diao, 2016) in the context of applying streaming approximation algorithm into finding soft assignment of labels in a semi-supervised manner to each vertex in a large-scale graph $G = (V, E, W)$, where V

is the set of vertices, E is the set of edges and $W = (w_{uv})$ is the edge weight matrix. Let V_l and V_u be the sets of labeled and unlabeled vertices, respectively. Let $n = |V|$, $n_l = |V_l|$ and $n_u = |V_u|$. We use diagonal matrix S to record the seeds, in which $s_{v,v} = 1$ if the node v is a seed. Let L represent the output label set of size m . Matrix $Y \in \mathbb{R}^{n \times m}$ records the training label distribution for the seeds where $Y_{vl} = 0$ for $v \in V_u$. Matrix $\hat{Y} \in \mathbb{R}^{n \times m}$ is the label distribution assignment matrix for all vertices. The graph-based SSL learns \hat{Y} by propagating the information Y on graph G . To obtain the label distribution \hat{Y} , we minimize the following convex objective function:

$$C(\hat{Y}) = \mu_1 \sum_{v \in V_l} s_{vv} \|\hat{Y}_v - Y_v\|_2^2 + \mu_2 \sum_{v \in V, u \in \mathcal{N}(v)} w_{vu} \|\hat{Y}_v - \hat{Y}_u\|^2 + \mu_3 \sum_{v \in V} \|\hat{Y}_v - U\|_2^2 \quad (3.47)$$

with a constraint:

$$\sum_{l=1}^L \hat{Y}_{vl} = 1 \quad (\forall v \in V) \quad (3.48)$$

where $\mathcal{N}(v)$ denotes the neighborhood of vertex v , U is the uniform prior distribution over all labels, and μ_1, μ_2, μ_3 are constant hyper-parameters. The objective function 3.47 satisfies the following:

- First term: For all labeled vertices (seeds), the label distribution should be close to the given label assignment.
- Second term: Close vertices should share similar labels.
- Third term: The label distribution should be close to the prior uniform distribution.

In addition, the objective function allows efficient iterative optimization algorithm that is repeated until convergence, in particular Jacobi iterative algorithm which defines the approximate solution at the $(i+1)$ -th iteration based on the solution of the i -th iteration:

$$\hat{Y}_{vl}^{(i+1)} = \frac{1}{M_{vl}} (\mu_1 s_{vv} Y_{vl} + \mu_2 \sum_{u \in \mathcal{N}(v)} w_{vu} \hat{Y}_{ul}^{(i)} + \mu_3 U_l) \quad (3.49)$$

$$M_{vl} = \mu_1 s_{vv} + \mu_2 \sum_{u \in \mathcal{N}(v)} w_{vu} + \mu_3 \quad (3.50)$$

where $U_l = 1/m$ which is the uniform distribution on label l . $\hat{Y}_{vl}^{(0)}$ is initialized with seed label weight Y_{vl} if $v \in V_l$, and uniform distribution $1/m$ if $v \in V_u$. This optimization method is called the EXPANDER algorithm.

3.7 Covariant Compositional Networks

Regardless of the specifics, the two major issues that graph learning methods need to grapple with are *invariance to permutations* and *capturing structure at multiple different scales*. Let A denote the adjacency matrix of \mathcal{G} , and suppose that we change the numbering of the vertices by applying a permutation σ . The adjacency matrix will then change to A' , with

$$A'_{i,j} = A_{\sigma^{-1}(i), \sigma^{-1}(j)}.$$

However, topologically, A and A' still represent the same graph. Permutation invariance means that the representation $\phi(\mathcal{G})$ learned or implied by our graph learning algorithm must be invariant to these transformations (see Fig. 3.1). Naturally, in the case of molecules, invariance is restricted to permutations that map each atom to another atom of the same type. While MPNNs have been very successful in applications and are an active field of research, they differ from classical CNNs in a fundamental way: the internal feature representations in CNNs are *equivariant* to such transformations of the inputs as translation and rotations (Cohen and Welling, 2016, 2017), the internal representations in MPNNs are fully invariant. This is a direct result of the fact that MPNNs deal with the permutation invariance issue in graphs simply by summing the messages coming from each neighbor. In this thesis, we argue that this is a serious limitation that restricts the representation power of MPNNs.

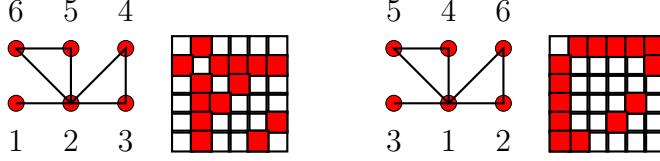


Figure 3.1: (a) A small graph G with 6 vertices and its adjacency matrix. (b) An alternative form G' of the same graph, derived from G by renumbering the vertices by a permutation $\sigma: \{1, 2, \dots, 6\} \mapsto \{1, 2, \dots, 6\}$. The adjacency matrices of G and G' are different, but topologically they represent the same graph. Therefore, we expect the feature map ϕ to satisfy $\phi(G) = \phi(G')$.

The multiscale property is equally crucial for learning molecular properties. For example, in the case of a protein, an ideal graph learning algorithm would represent \mathcal{G} in a manner that simultaneously captures structure at the level of individual atoms, functional groups, interactions between functional groups, subunits of the protein, and the protein’s overall shape. MPNNs are ultimately compositional (part-based) models, that build up the representation of the graph from the representations of a hierarchy of subgraphs. To address the covariance issue, we study the covariance behavior of such networks in general, introducing a new general class of neural network architectures, which we call *compositional networks* (comp-nets). One advantage of this generalization is that instead of focusing attention on the mechanics of how information propagates from node to node, it emphasizes the connection to convolutional networks, in particular, it shows that what is missing from MPNNs is essentially the analog of *steerability*. Steerability implies that the activations (feature vectors) at a given neuron must transform according to a specific representation (in the algebraic sense) of the symmetry group of its receptive field, in our case, the group of permutations, \mathbb{S}_m . In this thesis, we only consider the defining representation and its tensor products, leading to first, second, third etc. order *tensor activations*. We derive the general form of covariant tensor propagation in comp-nets, and find that each “channel” in the network corresponds to a specific way of contracting a higher order tensor to a lower order one. Note that here by *tensor activations* we mean not just that each activation is expressed as a multidimensional

array of numbers (as the word is usually used in the neural networks literature), but also that it transforms in a specific way under permutations, which is a more stringent criterion. The parameters of our covariant comp-nets are the entries of the mixing matrix that prescribe how these channels communicate with each other at each node.

3.7.1 Compositional networks

The idea of representing complex objects in terms of a hierarchy of parts has a long history in machine learning (Fischler and Elschlager, 1973; Ohta et al., 1978; Tu et al., 2005; Felzenszwalb and Huttenlocher, 2005; Zhu and Mumford, 2006; Felzenszwalb et al., 2010). We have recently introduced a general framework for encoding such part-based models in a special type of neural network, called *covariant compositional networks (CCNs)* (Kondor et al., 2018a). In this section, we show how the CCN formalism can be specialized to learning from graphs, specifically, the graphs of molecules. Our starting point is the following definition.

Definition 3.7.1. Let \mathcal{G} be the graph of a molecule made up of n atoms $\{e_1, \dots, e_n\}$. The **compositional neural network (comp-net)** corresponding to \mathcal{G} is a directed acyclic graph (DAG) \mathcal{N} in which each node (neuron) \mathbf{n}_i is associated with a subgraph \mathcal{P}_i of \mathcal{G} and carries an activation f_i . Moreover,

1. If \mathbf{n}_i is a leaf node, then \mathcal{P}_i is just a single vertex of \mathcal{G} , i.e., an atom $e_{\xi(i)}$, and the activation f_i , is some initial label l_i . In the simplest case, f_i is a “one-hot” vector that identifies what type of atom resides at the given vertex.
2. \mathcal{N} has a unique root node \mathbf{n}_r for which $\mathcal{P}_r = \mathcal{G}$, and the corresponding f_r represents the entire molecule.
3. For any two nodes \mathbf{n}_i and \mathbf{n}_j , if \mathbf{n}_i is a descendant of \mathbf{n}_j , then \mathcal{P}_i is a subgraph of \mathcal{P}_j , and

$$f_i = \Phi_i(f_{c_1}, f_{c_2}, \dots, f_{c_k}),$$

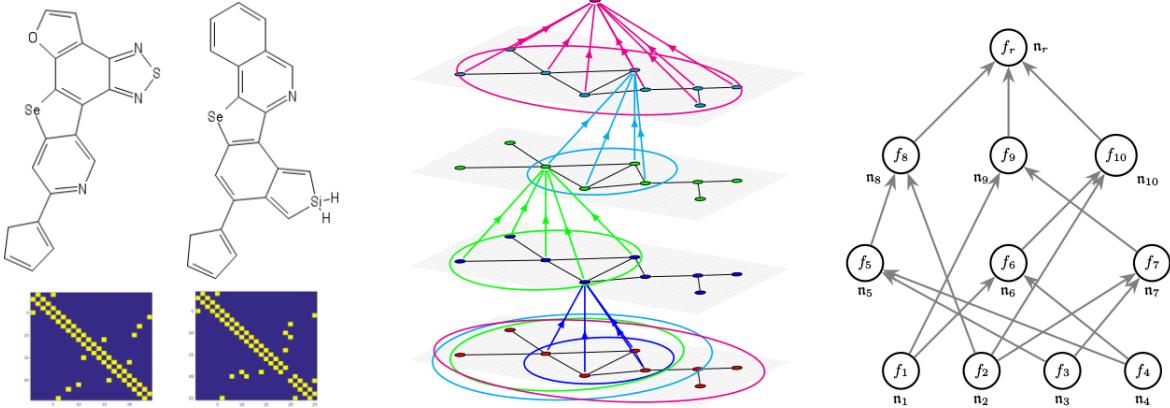


Figure 3.2: **Left:** Molecular graphs for $C_{18}H_9N_3OSSe$ and $C_{22}H_{15}NSeSi$ from the Harvard Clean Energy Project (HCEP) dataset with corresponding adjacency matrices. **Center and right:** The comp-net of a graph \mathcal{G} is constructed by decomposing \mathcal{G} into a hierarchy of subgraphs $\{\mathcal{P}_i\}$ and forming a neural network \mathcal{N} in which each “neuron” n_i corresponds to one of the \mathcal{P}_i subgraphs, and receives inputs from other neurons that correspond to smaller subgraphs contained in \mathcal{P}_i . The center pane shows how this can equivalently be thought of as an algorithm in which each vertex of \mathcal{G} receives and aggregates messages from its neighbors. To keep the figure simple, we only marked aggregation at a single vertex in each round (layer).

where f_{c_1}, \dots, f_{c_k} denote the activations of the children of n_i . Here, Φ_i is called the aggregation function of node i .

Note that we now have two separate graphs: the original graph \mathcal{G} , and the network \mathcal{N} that represents the “is a subgraph of” relationships between different subgraphs of \mathcal{G} . One of the fundamental ideas of this paper is that \mathcal{N} can be interpreted as a neural network, in which each node n_i is a “neuron” that receives inputs $(f_{c_1}, f_{c_2}, \dots, f_{c_k})$ and outputs the activation $f_i = \Phi_i(f_{c_1}, f_{c_2}, \dots, f_{c_k})$ (Figure 3.7.1, right). For now we treat the activations as vectors, but will soon generalize them to be tensors.

There is some freedom in how the system $\{\mathcal{P}_i\}$ of subgraphs is defined, but the default choice is $\{\mathcal{P}_i^\ell\}$, where \mathcal{P}_i^ℓ is the subgraph of vertices within a radius of ℓ of vertex i . In this case, $\mathcal{P}_i^0 = \{i\}$, \mathcal{P}^ℓ consists of the immediate neighbors of i , plus i itself, and so on. The aggregation

function is discussed in detail in Section 3.7.3.

Conceptually, comp-nets are closely related to convolutional neural networks (CNNs), which are the mainstay of neural networks in computer vision. In particular,

1. Each neuron in a CNN only aggregates information from a small set of neurons from the previous layer, similarly to how each node of a comp-net only aggregates information from its children.
2. The so-called *effective receptive fields* of the neurons in a CNN, i.e., the image patches for which each neuron is responsible for, form a hierarchy of nested sets similar to the hierarchy of $\{\mathcal{P}_i\}$ subgraphs.

In this sense, CNNs are a specific kind of compositional network, where the atoms are pixels. Because of this analogy, in the following we will sometimes refer to \mathcal{P}_i as the receptive field of neuron i , dropping the “effective” qualifier for brevity.

As mentioned above, an alternative popular framework for learning from graphs is message passing neural networks (MPNNs)(Gilmer et al., 2017). An MPNN operates in rounds: in each round $\ell = 1, \dots, L$, every vertex collects the labels of its immediate neighbors, applies a nonlinear function Ψ , and updates its own label accordingly. From the neural networks point of view, the rounds correspond to layers and the labels correspond to the f_i^ℓ activations. More broadly, the classic Weisfeiler–Lehman test of isomorphism follows the same logic (Weisfeiler and Lehman, 1968; Read and Corneil, 1977; Cai et al., 1992), and so does the related Weisfeiler–Lehman kernel, arguably the most successful kernel-based approach to graph learning (Shervashidze et al., 2011).

In the above mentioned base case when $\{\mathcal{P}_i^\ell\}$ is the collection of all subgraphs of \mathcal{G} of radii $\ell = 0, 1, 2, \dots$, a comp-net can also be thought of as a message passing algorithm: the messages

received by vertex i in round ℓ are the activations $\{f_{u_j}^{\ell-1} \mid u_j \in B(i, 1)\}$, where $B(i, 1)$ is the ball of radius one centered at i (note that this contains not just the neighbors of i , but also i itself). Conversely, MPNNs can be seen as comp-nets, where \mathcal{P}_i^ℓ is the subgraph defined by the receptive field of vertex i in round ℓ . A common feature of MPNNs, however, is that the Ψ aggregation functions that they employ are invariant to permuting their arguments. Most often, Ψ just *sums* all incoming messages and then applies a nonlinearity. This certainly guarantees that the final output of the network, $\phi(\mathcal{G}) = f_r$, will be permutation invariant. However, in the next section we argue that it comes at the price of a significant loss of representational power.

3.7.2 Permutation equivariant networks

One of the messages of the present paper is that *invariant* aggregation functions, of the type popularized by message passing neural networks, are *not* the most general way to build compositional models for compound objects, such as graphs. To understand why this is the case, once again, an analogy with image recognition is helpful. Classical CNNs face two types of basic image transformations: translations and rotations. With respect to translations (barring pooling, edge effects and other complications), CNNs behave in a quasi-invariant way, in the sense that if the input image is translated by an integer amount (t_x, t_y) , the activations in each layer $\ell = 1, 2, \dots, L$ translate the same way: the activation of neuron $\mathbf{n}_{i,j}^\ell$ is simply transferred to neuron $\mathbf{n}_{i+t_1, j+t_2}^\ell$, i.e., $f_{i+t_1, j+t_2}^{\ell\ell} = f_{i,j}^\ell$. This is the simplest manifestation of a well studied property of CNNs called *equivariance* (Cohen and Welling, 2016; Worrall et al., 2017).

With respect to rotations, however, the situation is more complicated: if we rotate the input image by, e.g., 90 degrees, not only will the part of the image that fell in the receptive field of neuron $\mathbf{n}_{i,j}^\ell$ move to the receptive field of a different neuron $\mathbf{n}_{-j,i}^\ell$, but the orientation of the

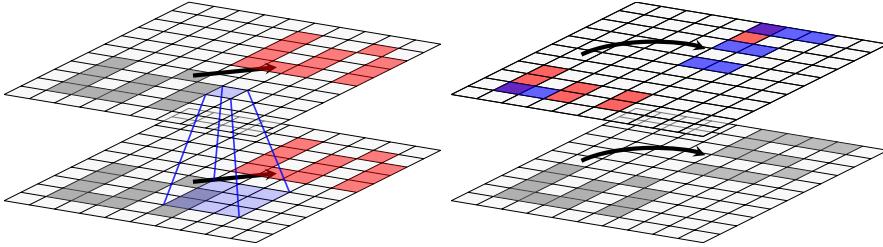


Figure 3.3: In a convolutional neural network if the input image is translated by some amount (t_1, t_2) , what used to fall in the receptive field of neuron $n_{i,j}^\ell$ is moved to the receptive field of $n_{i+t_1, j+t_2}^\ell$. Therefore, the activations transform in the very simple way $f'_{i+t_1, j+t_2}^\ell = f_{i,j}^\ell$. In contrast, rotations not only move the receptive fields around, but also permute the neurons in the receptive field internally, so the activations transform in a more complicated manner. The right hand figure shows that if the CNN has a horizontal filter (blue) and a vertical one (red), then after a rotation by 90 degrees, their activations are exchanged. In steerable CNNs, if (i, j) is moved to (i', j') by the transformation, then $f'_{i', j'}^\ell = R(f_{i,j}^\ell)$, where R is some fixed linear function of the rotation angle.

receptive field will also change. For example, features which were previously picked up by horizontal filters will now be picked up by vertical filters. Therefore, in general, $f'_{-j,i}^\ell \neq f_{i,j}^\ell$ (Figure 3.3).

It can be shown that one cannot construct a CNN for images that behaves in a quasi-invariant way with respect to both translations and rotations, unless every filter is directionless. It is, however, possible to construct a CNN in which the activations transform in a *predictable and reversible* way, $f'_{-j,i}^\ell = R(f_{i,j}^\ell)$, for some fixed function R . This phenomenon is called *steerability*, and has a significant literature in both classical signal processing (Freeman and Adelson, 1991; Simoncelli et al., 1992; Perona, 1995; Teo and Hel-Or, 1998; Manduchi et al., 1998) and the neural networks field (Cohen and Welling, 2017).

The situation in compositional networks is similar. The comp-net and message passing architectures that we have examined so far, by virtue of the aggregation function being symmetric in its arguments, are all *quasi-invariant* (with respect to permutations) in the sense that if \mathcal{N}

and \mathcal{N}' are two comp-nets for the same graph differing only in a reordering σ of the vertices of the underlying graph \mathcal{G} , and \mathbf{n}_i is a neuron in \mathcal{N} while \mathbf{n}'_j is the corresponding neuron in \mathcal{N}' , then $f_i = f'_j$ for any permutation $\sigma \in \mathbb{S}_n$.

Quasi-invariance amounts to asserting that the activation f_i at any given node must only depend on $\mathcal{P}_i = \{e_{j_1}, \dots, e_{j_k}\}$ as a *set*, and not on the internal ordering of the atoms e_{j_1}, \dots, e_{j_k} making up the receptive field. At first sight this seems desirable, since it is exactly what we expect from the overall representation $\phi(G)$. On closer examination, however, we realize that this property is potentially problematic, since it means that \mathbf{n}_i loses all information about which vertex in its receptive field has contributed what to the aggregate information f_i . In the CNN analogy, we can say that we have lost information about the *orientation* of the receptive field. In particular, if higher up in the network f_i is combined with some other feature vector f_j from a node with an overlapping receptive field, the aggregation process has no way of taking into account which parts of the information in f_i and f_j come from shared vertices and which parts do not (Figure 3.7.2).

The solution is to regard the \mathcal{P}_i receptive fields as *ordered sets*, and explicitly establish how f_i *co-varies* with the internal ordering of the receptive fields. To emphasize that henceforth the \mathcal{P}_i sets are ordered, we will use parentheses rather than braces to denote them.

Definition 3.7.2. Assume that \mathcal{N} is the comp-net of a graph \mathcal{G} , and \mathcal{N}' is the comp-net of the same graph but after its vertices have been permuted by some permutation σ . Given any $\mathbf{n}_i \in \mathcal{N}$ with receptive field $\mathcal{P}_i = (e_{p_1}, \dots, e_{p_m})$, let \mathbf{n}'_j be the corresponding node in \mathcal{N}' with receptive field $\mathcal{P}'_j = (e_{q_1}, \dots, e_{q_m})$. Assume that $\pi \in \mathbb{S}_m$ is the permutation that aligns the orderings of the two receptive fields, i.e., for which $e_{q_{\pi(a)}} = e_{p_a}$. We say that the comp-nets are **covariant to permutations** if for any π , there is a corresponding function R_π such that $f'_j = R_\pi(f_i)$.

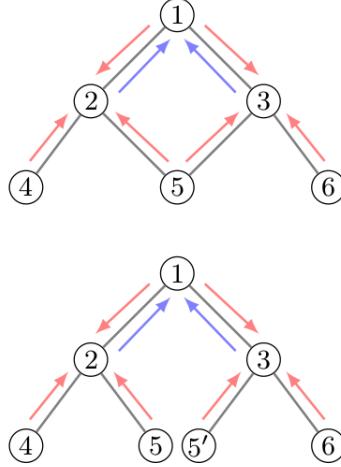


Figure 3.4: These two graphs are not isomorphic, but after a single round of message passing (red arrows), the labels (activations) at vertices 2 and 3 will be identical in both graphs. Therefore, in the second round of message passing vertex 1 will get the same messages in both graphs (blue arrows), and will have no way of distinguishing whether 5 and 5' are the same vertex or not.

To make the form of covariance prescribed by this definition more specific, we make the assumption that the $\{f \mapsto R_\pi(f)\}_{\pi \in \mathbb{S}_m}$ maps are *linear*. This allows us to treat them as matrices, $\{R_\pi\}_{\pi \in \mathbb{S}_n}$. Furthermore, linearity also implies that $\{R_\pi\}_{\pi \in \mathbb{S}_m}$ form a *representation* of \mathbb{S}_m in the group theoretic sense of the word (Serre, 1977) (this notion of representation should not be confused with the neural networks sense of representations of objects, as in “ f_i is a representation of \mathcal{P}_i ”).

The representation theory of symmetric groups is a rich subject that goes beyond the scope of the present paper (Sagan, 2001). However, there is one particular representation of \mathbb{S}_m that is likely familiar even to non-algebraists, the so-called *defining representation*, given by the $P_\pi \in \mathbb{R}^{n \times n}$ permutation matrices

$$[P_\pi]_{i,j} = \begin{cases} 1 & \text{if } \pi(j) = i \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to verify that $P_{\pi_2\pi_1} = P_{\pi_2}P_{\pi_1}$ for any $\pi_1, \pi_2 \in \mathbb{S}_m$, so $\{P_\pi\}_{\pi \in \mathbb{S}_m}$ is indeed a representation of \mathbb{S}_m . If the transformation rules of the f_i activations in a given comp-net are dictated by this representation, then each f_i must necessarily be a $|\mathcal{P}_i|$ dimensional vector, and intuitively each component of f_i carries information related to one specific atom in the receptive field, or the interaction of that specific atom with all the others collectively. We call this case first order permutation covariance.

Definition 3.7.3. We say that \mathbf{n}_i is a **first order covariant node** in a comp-net if under the permutation of its receptive field \mathcal{P}_i by any $\pi \in \mathbb{S}_{|\mathcal{P}_i|}$, its activation transforms as $f_i \mapsto P_\pi f_i$.

If $(R_g)_{g \in \mathfrak{G}}$ is a representation of a group \mathfrak{G} , the matrices $(R_g \otimes R_g)_{g \in \mathfrak{G}}$ also form a representation. Thus, one step up in the hierarchy from P_π -covariant comp-nets are $P_\pi \otimes P_\pi$ -covariant comp-nets, where the f_i feature vectors are now $|\mathcal{P}_i|^2$ dimensional vectors that transform under permutations of the internal ordering by π as $f_i \mapsto (P_\pi \otimes P_\pi)f_i$. If we reshape f_i into a matrix $F_i \in \mathbb{R}^{|\mathcal{P}_i| \times |\mathcal{P}_i|}$, then the action

$$F_i \mapsto P_\pi F_i P_\pi^\top$$

is equivalent to $P_\pi \otimes P_\pi$ acting on f_i . In the following, we will prefer this more intuitive matrix view, since it makes it clear that feature vectors that transform this way express *relationships* between the different constituents of the receptive field. Note, in particular, that if we define $A \downarrow_{\mathcal{P}_i}$ as the restriction of the adjacency matrix to \mathcal{P}_i (i.e., if $\mathcal{P}_i = (e_{p_1}, \dots, e_{p_m})$) then $[A \downarrow_{\mathcal{P}_i}]_{a,b} = A_{p_a, p_b}$, then $A \downarrow_{\mathcal{P}_i}$ transforms exactly as F_i does in the equation above.

Definition 3.7.4. We say that \mathbf{n}_i is a **second order covariant node** in a comp-net if under the permutation of its receptive field \mathcal{P}_i by any $\pi \in \mathbb{S}_{|\mathcal{P}_i|}$, its activation transforms as $F_i \mapsto P_\pi F_i P_\pi^\top$.

Taking the pattern further lets us define third, fourth, and general, k 'th order nodes, in which the activations are k 'th order tensors, transforming under permutations as $F_i \mapsto F'_i$,

where

$$[F'_i]_{j_1, \dots, j_k} = [P_\pi]_{j_1, j'_1} [P_\pi]_{j_2, j'_2} \cdots [P_\pi]_{j_k, j'_k} [F_i]_{j'_1, \dots, j'_k}. \quad (3.51)$$

Here and in the following, for brevity, we use the Einstein summation convention, whereby any dummy index that appears twice on the right hand side of an equation is automatically summed over.

In general, we will call any quantity which transforms under permutations according to feq: kth action a *k'th order P-tensor*. Saying that a given quantity is a *P-tensor* then not only means that it is representable by an $m \times m \times \dots \times m$ array, but also that this array transforms in a specific way under permutations.

Since scalars, vectors and matrices can be considered 0th, 1st and 2nd order tensors, the following definition covers both quasi-invariance and first and second order covariance as special cases. To unify notation and terminology, in the following we will always talk about *feature tensors* rather than *feature vectors*, and denote the activations as F_i rather than f_i .

Definition 3.7.5. We say that \mathbf{n}_i is a **k'th order covariant node** in a comp-net if the corresponding activation F_i is a *k'th order P-tensor*, i.e., it transforms under permutations of \mathcal{P}_i according to feq: kth action, or the activation is a sequence of d separate *P-tensors* $F_i^{(1)}, \dots, F_i^{(d)}$ corresponding to d distinct channels.

A **covariant compositional network (CCN)** is a comp-net in which each node's activation is covariant to permutations in the above sense. Hence we can talk about first, second, third, etc. order CCNs (CCN 1D, 2D, ...). In the first few layers of the network, however, the order of the nodes might be lower (Figure 3.7.2).

The real significance of covariance, both here and in classical CNNs, is that while it allows for a richer internal representation of the data than fully invariant architectures, the final output

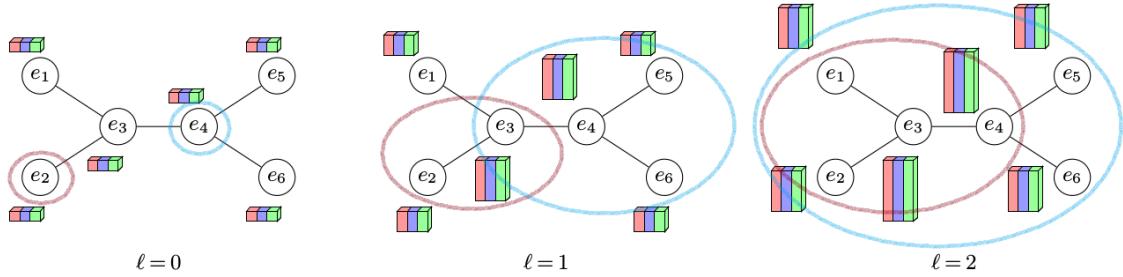


Figure 3.5: Feature tensors in a first order CCN for ethylene (C_2H_4) assuming three channels (red, green, and blue). Vertices e_1, e_2, e_5, e_6 are hydrogen atoms, while vertices e_3, e_4 are carbons. Edge (e_3, e_4) is a double bond, the other edges are single bonds. (a) At the input layer, the receptive field for each atom is just the atom itself, so the feature matrix has size 1×3 . (b) At level $\ell = 1$ the size of the receptive field of each atom grows depending on the local topology. The receptive fields for each hydrogen atoms grows to include its neighboring carbon atom, resulting in a feature tensor of size 2×3 ; the receptive fields of the carbon atoms grow to include four atoms each, and therefore have size 4×3 . (c) At layer $\ell = 2$, the receptive fields of the carbons will include every atom in the molecule, while the receptive fields of the hydrogens will only be of size 4.

of the network can still easily be made invariant. In covariant comp-nets this is achieved by collapsing the input tensors of the root node n_r at the top of the network into invariant scalars, for example by computing their sums and traces (reducing them to zeroth order tensors), and outputting permutation invariant combinations of these scalars, such as their sum.

3.7.3 Covariant aggregation functions

It remains to explain how to define the Φ aggregation functions so as to guarantee covariance. Specifically, we show how to construct Φ such that if the F_{c_1}, \dots, F_{c_k} inputs of a given node n_a at level ℓ are covariant k 'th order P -tensors, then $F_a = \Phi(F_{c_1}, \dots, F_{c_k})$ will also be a k 'th order P -tensor. The aggregation function that we define consists of five operations executed in sequence: promotion, stacking, reduction, mixing, and an elementwise nonlinear transform. Practically relevant CCNs tend to have multiple channels, so each F_{c_i} is actually a sequence of d separate P -tensors $F_{c_i}^{(1)}, \dots, F_{c_i}^{(d)}$. However, except for the mixing step,

each channel behaves independently, so for simplicity, for now we drop the channel index. Figs. 3.6 and 3.11 shows the aggregation process and neural activations on the second order CCN.

Promotion

Each child tensor F_{c_i} captures information about a different receptive field \mathcal{P}_{c_i} , so before combining them we must “promote” each F_{c_i} to a $|\mathcal{P}_a| \times \dots \times |\mathcal{P}_a|$ tensor \tilde{F}_{c_i} , whose dimensions are indexed by the vertices of \mathcal{P}_a rather than the vertices in each \mathcal{P}_{c_i} . Assuming that $\mathcal{P}_{c_i} = (e_{q_1}, \dots, e_{q_{|\mathcal{P}_{c_i}|}})$ and $\mathcal{P}_b = (e_{p_1}, \dots, e_{p_{|\mathcal{P}_b|}})$, this is done by defining a $|\mathcal{P}_a| \times |\mathcal{P}_{c_i}|$ indicator matrix

$$\chi_{i,j}^{c_i \rightarrow a} = \begin{cases} 1 & \text{if } q_j = p_i \\ 0 & \text{otherwise,} \end{cases}$$

and setting

$$[\tilde{F}_{c_i}]_{j_1, \dots, j_k} = [\chi^{c_i \rightarrow b}]_{j_1, j'_1} [\chi^{c_i \rightarrow b}]_{j_2, j'_2} \dots [\chi^{c_i \rightarrow b}]_{j_k, j'_k} [F_{c_i}]_{j'_1, \dots, j'_k},$$

where, once again, Einstein summation is in use, so summation over j'_1, \dots, j'_k is implied. Effectively, the promotion step aligns all the child tensors by permuting the indices of F_{c_i} to conform to the ordering of the atoms in \mathcal{P}_a , and padding with zeros where necessary.

Stacking

Now that the promoted tensors $\tilde{F}_{c_1}, \dots, \tilde{F}_{c_s}$ all have the same shape, they can be stacked to form a $|\mathcal{P}_a| \times \dots \times |\mathcal{P}_a|$ dimensional $k+1$ ’th order tensor T (see figure 3.7 for an example),

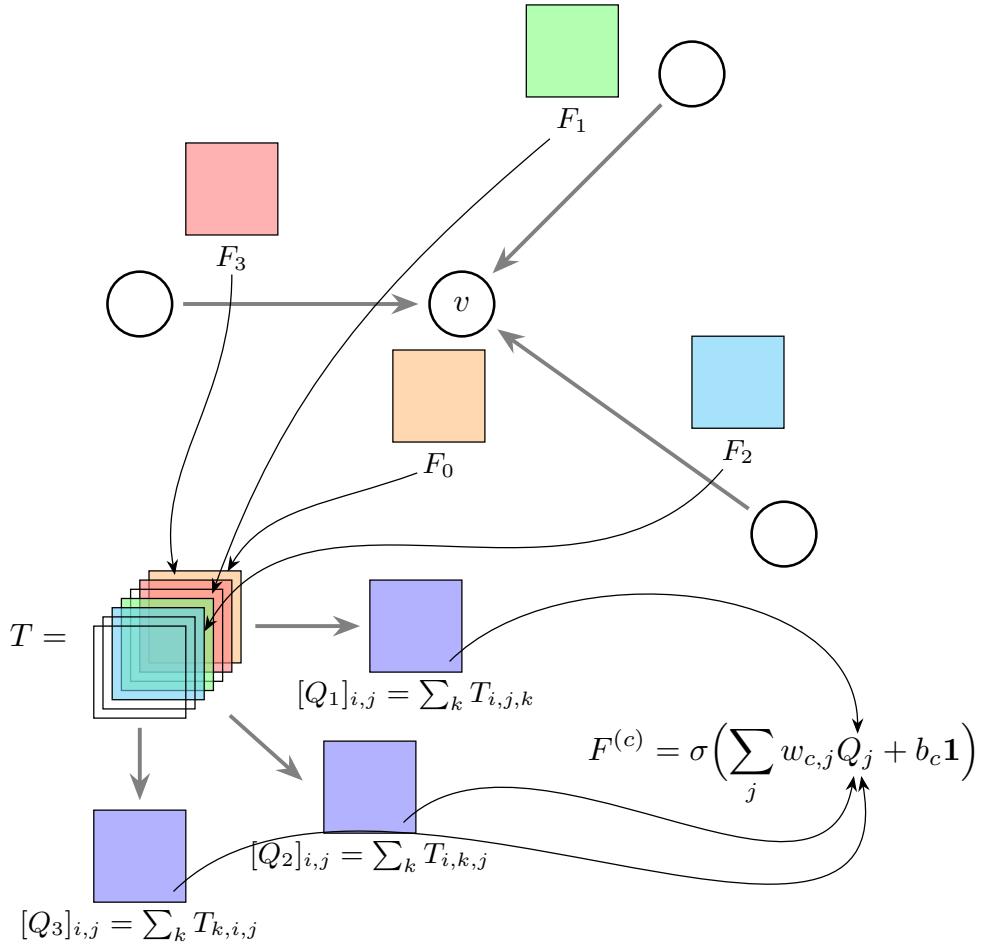
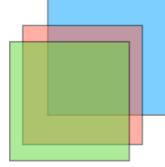


Figure 3.6: Schematic of the aggregation process at a given vertex v of \mathcal{G} in a second order CCN not involving tensor products with $A \downarrow \mathcal{P}_b$ and assuming a single input channel. Feature tensors F_0, \dots, F_3 are collected from the neighbors of v as well as from v itself, promoted, and stacked to form a third order tensor T . In this example we compute just three reductions Q_1, Q_2, Q_3 . These are then combined them with the $w_{c,j}$ weights and passed through the σ nonlinearity to form the output tensors $(F^{(1)}, F^{(2)}, F^{(3)})$. For simplicity, in this figure the “ ℓ ” and “ a ” indices are suppressed.



$$C_{i,j,k} = A_{i,j}^{(k)}$$

Figure 3.7: Stacking second order tensors (matrices) $A^{(k)}$ s into a third order tensor C .

with

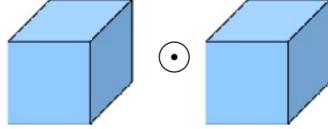
$$T_{j_0, \dots, j_k} = \begin{cases} [\tilde{F}_{c_i}]_{j_0, \dots, j_k} & \text{if } \mathcal{P}_{c_i} \text{ is the subgraph} \\ & \text{centered at } e_{p_{j_0}}, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that T itself transforms as a P -tensor of order $(k+1)$.

We may also add additional information to T by taking its Hadamard/element-wise product or tensor product with another P -tensor (see figures 3.8 and 3.9). In particular, to explicitly add information about the local topology, we may tensor multiply T by $[A \downarrow \mathcal{P}_a]_{i,j} = A_{e_{p_i}, e_{p_j}}$, the restriction of the adjacency matrix to \mathcal{P}_a . This will give an order $(k+3)$ tensor $S = T \otimes A \downarrow \mathcal{P}_a$. Otherwise, we just set $S = T$. Note that in most other graph neural networks, the topology of the underlying graph is only accounted for implicitly, by the pattern in which different activations are combined. Being able to add the local adjacency matrix explicitly greatly extends the representational power of CCNs.

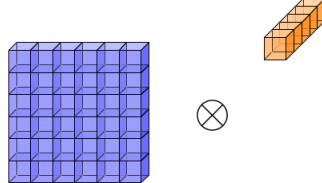
Reduction

Stacking and the optional tensor product increase the order of our tensors from k to $k+1$ or $k+3$. We reduce this to k again by a combination of generalized tensor contractions, more accurately called *tensor reductions* (see figure 3.10). For example, one way to drop the rank



$$C_{i,j,k} = A_{i,j,k} B_{i,j,k}$$

Figure 3.8: Hadamard/element-wise product preserves covariance.



$$C_{i,j,k} = A_{i,j} B_k$$

Figure 3.9: Tensor product between a first order tensor (vector) B with a second order tensor (local adjacency matrix) A results into a third order tensor C .

of S from $k+3$ to k is to sum out three of its indices,

$$Q_{i_1, \dots, i_k} = \sum_{i_{\alpha_1}, i_{\alpha_2}, i_{\alpha_3}} S_{i_1, \dots, i_k}.$$

Note that while this notation does not make it explicit, $i_{\alpha_1}, i_{\alpha_2}$ and i_{α_3} must be removed from amongst the indices of Q . Another way to drop the rank is to *contract* over three indices,

$$Q_{i_1, \dots, i_k} = \sum_{i_{\alpha_1}, i_{\alpha_2}, i_{\alpha_3}} S_{i_1, \dots, i_k} \delta_{i_{\alpha_1}, i_{\alpha_2}, i_{\alpha_3}},$$

where $\delta_{i_{\alpha_1}, i_{\alpha_2}, i_{\alpha_3}}$ is the generalized Kronecker symbol. A third way is to contract over two indices and sum over one index, and so on. The crucial fact is that result of each of these tensor operations is still a covariant P -tensor.

In general, the number of different ways that an order $k+q$ tensor S can be reduced to order

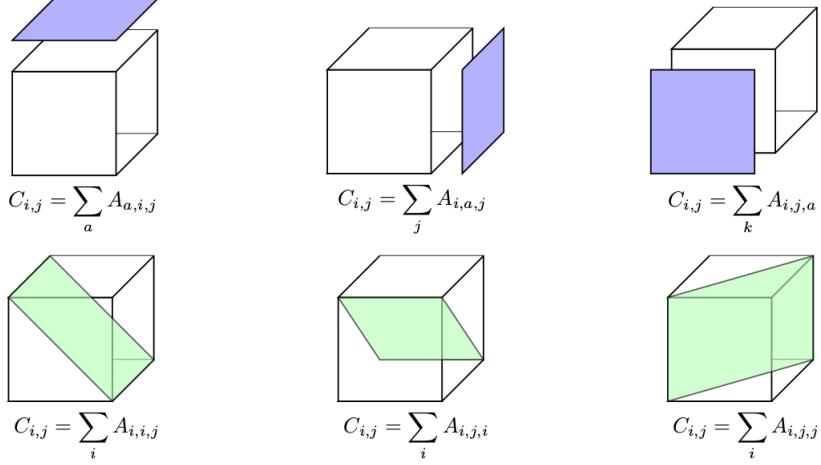


Figure 3.10: There are six different ways of covariantly reducing (contracting) a third order tensor A to a second order tensor (matrix) C : by either projecting into one side or taking the diagonal.

k depends on both q and k . For example, when $k=2$ and $q=3$, there are 50 different possible tensor reductions (excluding diagonals). In contrast, when $q=1$ (i.e., we are not multiplying by the adjacency matrix), we only have $k+1$ possibilities, corresponding to summing S over each of its dimensions. No matter which case we are in, however, and how many contractions Q_1, \dots, Q_d our network actually computes (in our experiments using second order nodes, we compute 18 different ones), what is important is that the resulting order k tensors satisfy the P -tensor property.

Mixing with learnable weights

The reduction step can potentially produce quite a large number of order k tensors, Q_1, \dots, Q_r . We reduce this number by linearly *mixing* Q_1, \dots, Q_r , i.e., taking $d' < r$ linear combinations of the form

$$\tilde{Q}^{(i)} = \sum_{j=1}^r w_{i,j}^{(\ell)} Q_j. \quad (3.52)$$

This is again a covariant operation, and the mixing weights are the actual learnable parameters of our neural network.

It is natural to interpret $\tilde{Q}^{(1)}, \dots, \tilde{Q}^{(d')}$ as d' separate channels, in neural network terminology. Recall that we also allow the input tensors to have multiple channels, but up to now the corresponding index has been suppressed. The mixing step is the point where these channels are all allowed to interact, so eq: mixing becomes

$$\tilde{Q}^{(c)} = \sum_{c',j} w_{c,c',j}^{(\ell)} Q_j^{(c')}, \quad (3.53)$$

and the learnable weights of the network at each level form a third order tensor $W_\ell = (w_{c,c',j}^{(\ell)})_{c,c',j}$.

The channel index c is not to be confused with c_1, \dots, c_k denoting the children of node a . Equation 3.52 is the main mechanism whereby CCNs are able to learn increasingly complex topological features as we move up the network.

Nonlinearity

Finally, to get the actual activation of our neuron n_a , we add an additive bias $b_{\ell,c}$, and an elementwise nonlinearity σ (specifically, the ReLU operator $\sigma(x) = \max\{0, x\}$), as is standard in neural networks. Thus, ultimately, the output of the aggregation function is the collection of P -covariant tensors

$$F_a^{(c)} = \sigma \left[\sum_{c'=1}^d \sum_{j=1}^r w_{c,c',j}^{(\ell)} Q_j^{(c')} + b_{\ell,c} \mathbb{1} \right] \quad (3.54)$$

with $c \in \{1, 2, \dots, d'\}$. As usual in neural networks, the W_ℓ weight tensors are learned by backpropagation and some form of stochastic gradient descent.

3.7.4 Architecture

In this section, we will describe the architecture of our permutation equivariant networks that is a generalization of previous works with an extension to higher-order representations. Recent works on graph neural networks (Duvenaud et al., 2015; Kipf and Welling, 2017; Li

et al., 2016a; Gilmer et al., 2017) can all be seen as instances of *zeroth order message passing* where each vertex representation is a vector (first order tensor) of c channels in which each channel is represented by a scalar (zeroth order P-tensor). This results in the loss of certain structural information during the message aggregation, and the network loses the ability to learn topological information of the graph’s multiscale structure.

Our architecture represents generalized vertex representations with higher-order tensors which can retain this structural information. There is significant freedom in the choice of this tensor structure, and we now explore two examples, corresponding to the tensor structures, which we call “first order CCN” and “second order CCN”, respectively.

We start with an input graph $G = (V, E)$ and construct a network with $L + 1$ levels, indexed from 0 (input level) to L (top level). Initially, each vertex v is associated with an input feature vector $l_v \in \mathbb{R}^c$ where c denotes the number of channels. The receptive field of vertex v at level ℓ is denoted by \mathcal{P}_v^ℓ and is defined recursively as follows:

$$\mathcal{P}_v^\ell \triangleq \begin{cases} \{v\}, & \ell = 0 \\ \bigcup_{(u,v) \in E} \mathcal{P}_u^\ell, & \ell = 1, \dots, L \end{cases} \quad (3.55)$$

The vertex representation of vertex v at level ℓ is denoted by a feature tensor F_v^ℓ . In zeroth order message passing, $F_v^\ell \in \mathbb{R}^c$ is a vector of c channels. Let N be the number of vertices in \mathcal{P}_v^ℓ . In the first order CCN, each vertex is represented by a matrix (second order tensor) $F_v^\ell \in \mathbb{R}^{N \times c}$ in which each row corresponds to a vertex in the receptive field \mathcal{P}_v^ℓ , and each channel is represented by a vector (first order P-tensor) of size N . In the second order CCN, F_v^ℓ is promoted into a third order tensor of size $N \times N \times c$ in which each channel has a second order representation (second order P-tensor). In general, we can imagine a series of feature

tensors of increasing order for higher order message passing. Note that the components corresponding to the channel index does not transform as a tensor, whereas the remaining indices do transform as a P-tensor. The tensor F_v^ℓ transforms in a *covariant* way with respect to the permutation of the vertices in the receptive field \mathcal{P}_v^ℓ .

Now that we have established the structure of the high order representations of the vertices at each site, we turn to the task of constructing the aggregation function Φ from one level to another. The key to doing this in a way that preserves covariance is to “promote-stack-reduce” the tensors as one traverses the network at each level.

We start with the promotion step. Recall that we want to accumulate information at higher levels based upon the receptive field of a given vertex. However, it is clear that not all vertices in the receptive field have the same size tensors. To account for this, we use an index function χ that ensures all tensors are the same size by padding with zeros when necessary. At level ℓ , given two vertices v and w such that $\mathcal{P}_w^{\ell-1} \subseteq \mathcal{P}_v^\ell$, the permutation matrix $\chi_\ell^{w \rightarrow v}$ of size $|\mathcal{P}_v^\ell| \times |\mathcal{P}_w^{\ell-1}|$ is defined as in section 3.7.3. In CCN 1D & 2D, the resizing is done by (broadcast) matrix multiplication $\chi \cdot F_w^{\ell-1}$ and $\chi \times F_w^{\ell-1} \times \chi^T$ where $\chi = \chi_\ell^{w \rightarrow v}$, respectively. Denote the resized tensor as $F_{w \rightarrow v}^\ell$. (See step 7 in algorithm 15.) This promotion is done for all tensors of every vertex in the receptive field, and stacked/concatenated into a tensor one order higher. Notice that the stacked index has the same size as the receptive field. From here, as in CCN 2D, we can compute the *tensor product* of this higher order tensor with the restricted adjacency matrix (subject to the receptive field) and obtain an even higher order tensor. Finally, we can reduce the higher order tensor down to the expected size of the vertex representation using the tensor contractions discussed in section 3.7.3.

We include all possible tensor contractions, which introduces additional channels. To avoid

an exponential explosion in the number of channels with deep networks, we use a learnable set of weights that reduces the number of channels to a fixed number c . These weights are learnable through backpropagation. To complete our construction of Φ , this tensor is passed through an element-wise nonlinear function Υ such as a ReLU to form the feature tensor for a given vertex at the next level. (See steps 4 and 9 in algorithm 15.)

Finally, at the output of the network, we again reduce the vertex representations F_v^ℓ into a vector of channels $\Theta(F_v^\ell) = F_v^\ell \downarrow_{i_1, \dots, i_p}$ where i_1, \dots, i_p are the non-channel indices. We sum up all the reduced vertex representations of a graph to get a single vector which we use as the graph representation. This final graph representation can then be used for regression or classification with a fully connected layer. In addition, we can construct a richer graph representation by concatenating the shrunk representation at each level. (See steps 12, 13 and 14 in algorithm 15.)

The development of higher order CCNs require efficient tensor algorithms to successively train the network. A fundamental roadblock we face in implementing CCNs is that fifth or sixth order tensors are often too large to be held in memory. To address this challenge, we do not construct the tensor product explicitly. Instead we introduce a *virtual indexing system* for a *virtual tensor* that computes the elements of tensor only when needed given the indices. This allows us to implement the tensor contraction operations efficiently with GPUs on virtual tensors.

For example, consider the operations in step 8 in algorithm 15. This requires performing contractions over several indices on the two inputs $\mathcal{F} = \{F_{w \rightarrow v}^\ell | w \in \mathcal{P}_v^\ell\}$, in which $\mathcal{F}_{i_1} = F_{w_{i_1} \rightarrow v}^\ell$ is of size $|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times c$, and $\mathcal{A} = A \downarrow_{\mathcal{P}_v^\ell}$. One naive strategy would be to stack all tensors in \mathcal{F} into a new object and then directly compute the tensor product with \mathcal{A} to form a

sixth order tensor, given by a tuple of indices $(i_1, i_2, i_3, i_4, i_5, i_6)$. Instead, the corresponding tensor element is computed on-the-fly through simple multiplication:

$$\mathcal{T}_{i_1, i_2, i_3, i_4, i_5, i_6} = (\mathcal{F}_{i_1})_{i_2, i_3, i_6} \cdot \mathcal{A}_{i_4, i_5} \quad (3.56)$$

where i_6 is the channel index. In this case, \mathcal{T} is 5th order (6th order if we consider the channel index), and can be contracted down to second order in the following ways:

1. The **1+1+1** case contracts \mathcal{T} in the form $\mathcal{T}_{i_1, i_2, i_3, i_4, i_5} \delta^{i_{a1}} \delta^{i_{a2}} \delta^{i_{a3}}$, i.e., it projects \mathcal{T} down along 3 of its 5 dimensions. This can be done in $\binom{5}{3} = 10$ ways.
2. The **1+2** case contracts \mathcal{T} in the form $\mathcal{T}_{i_1, i_2, i_3, i_4, i_5} \delta^{i_{a1}} \delta^{i_{a2}, i_{a3}}$, i.e., it projects \mathcal{T} along one dimension, and contracts it along two others. This can be done in $3 \binom{5}{3} = 30$ ways.
3. The **3** case is a single 3-fold contraction $\mathcal{T}_{i_1, i_2, i_3, i_4, i_5} \delta^{i_{a1}, i_{a2}, i_{a3}}$. This can be done in $\binom{5}{3} = 10$ ways.

Totally, we have 50 different contractions that result in 50 times more channels. In our experiments of CCN 2D, for efficiency, we only implement 18 different contractions such that each contraction results in a $|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times c$ tensor. The result of step 8 is \bar{F}_ℓ^v with 18 times more channels.

Algorithm CCN 1D is described in pseudocode 14. Figure 3.7.2 shows a visualization of CCN 1D's tensors on C_2H_4 molecular graph. Vertices e_3 and e_4 are carbon (C) atoms, and vertices e_1, e_2, e_5 and e_6 are hydrogen (H) atoms. Edge (e_3, e_4) is a double bond (C, C) between two carbon atoms. All other edges are single bonds (C, H) between a carbon atom and a hydrogen atom. In the bottom layer $\ell = 0$, the receptive field of every atom e only contains itself, thus its representation F_e^0 is a tensor of size $1 \times c$ where c is the number of channels. In the first layer $\ell = 1$, the receptive field of a hydrogen atom contains itself and

the neighboring carbon atom (i.e., $\mathcal{P}_{e_1}^1 = \{e_1, e_3\}$), thus tensors for hydrogen atoms are of size $2 \times c$. Meanwhile, the receptive field of a carbon atom contains itself, the another carbon and two other neighboring hydrogens (i.e., $\mathcal{P}_{e_3}^1 = \{e_1, e_2, e_3, e_4\}$) and $\mathcal{P}_{e_4}^1 = \{e_3, e_4, e_5, e_6\}$), thus $F_{e_3}^1, F_{e_4}^1 \in \mathbb{R}^{4 \times c}$. In all later layers denoted $\ell = \infty$, the receptive field of every atom contains the whole graph (in this case, 6 vertices in total), thus $F_e^\infty \in \mathbb{R}^{6 \times c}$.

Algorithm 14 First-order CCN

```

1: Input:  $G, l_v, L$ 
2: Parameters: Matrices  $W_0 \in \mathbb{R}^{c \times c}, W_1, \dots, W_L \in \mathbb{R}^{(2c) \times c}$  and biases  $b_0, \dots, b_L$ . For CCN
   1D, we only implement 2 tensor contractions.
3:  $F_v^0 \leftarrow \Upsilon(W_0 l_v + b_0 \mathbb{1})$  ( $\forall v \in V$ )
4: Reshape  $F_v^0$  to  $1 \times c$  ( $\forall v \in V$ )
5: for  $\ell = 1, \dots, L$  do
6:   for  $v \in V$  do
7:      $F_{w \rightarrow v}^\ell \leftarrow \chi \times F_w^{\ell-1}$  where  $\chi = \chi_{w \rightarrow v}^\ell$  ( $\forall w \in \mathcal{P}_v^\ell$ )
8:     Concatenate the promoted tensors in  $\{F_{w \rightarrow v}^\ell | w \in \mathcal{P}_v^\ell\}$  and apply 2 tensor contrac-
       tions that results in  $\bar{F}_v^\ell \in \mathbb{R}^{|\mathcal{P}_v^\ell| \times (2c)}$ .
9:      $F_v^\ell \leftarrow \Upsilon(\bar{F}_v^\ell \times W_\ell + b_\ell \mathbb{1})$ 
10:    end for
11:   end for
12:    $F^\ell \leftarrow \sum_{v \in V} \Theta(F_v^\ell)$  ( $\forall \ell$ )
13: Graph feature  $F \leftarrow \bigoplus_{\ell=0}^L F^\ell \in \mathbb{R}^{(L+1)c}$ 
14: Use  $F$  for downstream tasks.

```

Algorithm CCN 2D is described in pseudocode 15. Figure 3.11 shows a visualization of CCN 2D's tensors on C₂H₄ molecular graph. In the bottom layer $\ell = 0$, $|\mathcal{P}_e^0| = 1$ and $F_e^0 \in \mathbb{R}^{1 \times 1 \times c}$ for every atom e . In the first layer $\ell = 1$, $|\mathcal{P}_e^1| = 2$ and $F_e^1 \in \mathbb{R}^{2 \times 2 \times c}$ for hydrogen atom $e \in \{e_1, e_2, e_5, e_6\}$, and for carbon atoms $|\mathcal{P}_{e_3}^1| = |\mathcal{P}_{e_4}^1| = 4$ and $F_{e_3}^1, F_{e_4}^1 \in \mathbb{R}^{4 \times 4 \times c}$. In all other layers $\ell = \infty$, $\mathcal{P}_e^\infty \equiv V$ and $F_e^\infty \in \mathbb{R}^{6 \times 6 \times c}$ ($\forall e$).

Figure 3.12 shows the difference among zeroth, first and second order message passing (see

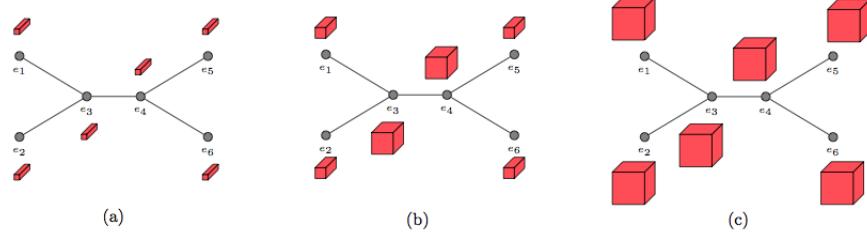


Figure 3.11: Neural activations of the second order CCN on C_2H_4 molecular graph: (a) initialization, (b) after 1 iteration, and (c) after 2 iterations.

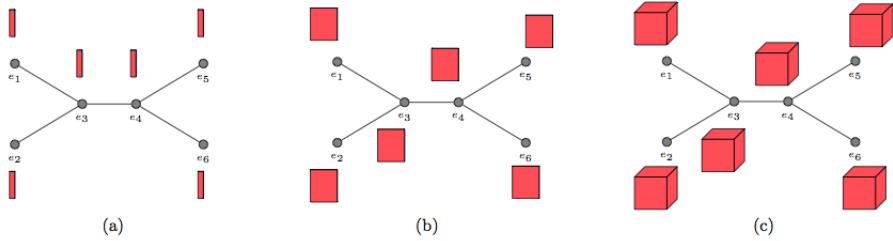
Algorithm 15 Second-order CCN

```

1: Input:  $G, l_v, L$ 
2: Parameters: Matrices  $W_0 \in \mathbb{R}^{c \times c}, W_1, \dots, W_L \in \mathbb{R}^{(18c) \times c}$  and biases  $b_0, \dots, b_L$ .
3:  $F_v^0 \leftarrow \Upsilon(W_0 l_v + b_0 \mathbb{1})$  ( $\forall v \in V$ )
4: Reshape  $F_v^0$  to  $1 \times 1 \times c$  ( $\forall v \in V$ )
5: for  $\ell = 1, \dots, L$  do
6:   for  $v \in V$  do
7:      $F_{w \rightarrow v}^\ell \leftarrow \chi \times F_w^{\ell-1} \times \chi^T$  where  $\chi = \chi_{w \rightarrow v}^\ell$  ( $\forall w \in \mathcal{P}_v^\ell$ )
8:     Apply virtual tensor contraction algorithm with inputs  $\{F_{w \rightarrow v}^\ell | w \in \mathcal{P}_v^\ell\}$  and the
       restricted adjacency matrix  $A \downarrow_{\mathcal{P}_v^\ell}$  to compute  $\bar{F}_v^\ell \in \mathbb{R}^{|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times (18c)}$ .
9:      $F_v^\ell \leftarrow \Upsilon(\bar{F}_v^\ell \times W_\ell + b_\ell \mathbb{1})$ 
10:    end for
11:  end for
12:   $F^\ell \leftarrow \sum_{v \in V} \Theta(F_v^\ell)$  ( $\forall \ell$ )
13: Graph feature  $F \leftarrow \bigoplus_{\ell=0}^L F^\ell \in \mathbb{R}^{(L+1)c}$ 
14: Use  $F$  for downstream tasks.

```

Figure 3.12: Zeroth, first and second order message passing



from left to right) with layer $\ell \geq 2$. In the zeroth order, the vertex representation is always a vector of c channels. In the first and second order, the vertex representation is a matrix of size $6 \times c$ or a 3rd order tensor of size $6 \times 6 \times c$ in which each channels is represented by a vector of length 6 or a matrix of size 6×6 , respectively. With higher orders, CCNs can capture more topological information.

3.7.5 Learning to estimate Density Functional Theory calculation

Density functional theory (Hohenberg and Kohn, 1964) (DFT) is the workhorse of modern quantum chemistry, due to its ability to calculate many properties of molecular systems with high accuracy. However, this accuracy comes at a significant computational cost, generally making DFT too costly for applications such as chemical search and drug screening, which may involve hundreds of thousands of compounds. Methods that help overcome this limitation could lead to rapid developments in biology, medicine, and materials engineering.

Recent advances in machine learning, specifically, deep learning (LeCun et al., 2015), combined with the appearance of large datasets of molecular properties obtained both experimentally and theoretically (Ramakrishnan et al., 2014; Hachmann et al., 2011; Kirklin et al., 2015) present an opportunity to *learn* to predict the properties of compounds from their chemical structure rather than computing them explicitly with DFT. A machine learning model could allow for rapid and accurate exploration of huge molecular spaces to find suit-

able candidates for a desired molecule.

Central to any machine learning technique is the choice of a suitable set of descriptors, or *features* used to parametrize and describe the input data. A poor choice of features will limit the expressiveness of the learning architecture and make accurate predictions impossible. On the other hand, providing too many features may make training difficult, especially when training data is limited. Hence, there has been a significant amount of work on designing good features for molecular systems. Predictions of energetics based on molecular geometry have been explored extensively, using a variety of parametrizations (Hansen et al., 2015; Huang and von Lilienfeld, 2016). This includes bond types and/or angles (Faber et al., 2017), radial distance distributions (von Lilienfeld et al., 2015), the Coulomb matrix and related structures (Rupp et al., 2012; Hansen et al., 2013; Montavon et al., 2013), the Smooth Overlap of Atomic Positions (SOAP) (Bartók et al., 2013; Bartók et al., 2017), permutation-invariant distances (Ferré et al., 2015), symmetry functions for atomic positions (Behler and Parrinello, 2007; Behler, 2011), Moment Tensor Potentials (Shapeev, 2016), and Scattering Networks (Hirn et al., 2017).

Recently, the problem of learning from the structure of chemical bonds alone, i.e., the molecular graph, has attracted a lot of interest, especially in light of the appearance of a series of neural network architectures designed specifically for learning from graphs (Bruna et al., 2014; Duvenaud et al., 2015; Kearnes et al., 2016; Bronstein et al., 2017; Schütt et al., 2017; Gilmer et al., 2017; Schütt et al., 2017). Much of the success of these architectures stems from their ability to pick up on structure in the graph at multiple different scales, while satisfying the crucial requirement that the output be invariant to permutations of the vertices (which, in molecular learning, correspond to atoms). However, as will explain, the specific way that most of these architectures ensure permutation invariance still limits their representational

power.

Importantly, CCNs are based on explicitly decomposing compound objects (in our case, molecular graphs) into a hierarchy of subparts (subgraphs), offering a versatile framework that is ideally suited to capturing the multiscale nature of molecular structures from functional groups through local structure to overall shape. In a certain sense, the resulting models are the neural networks analog of coarse graining. In addition, CCNs offer a more nuanced take on permutation invariance than other graph learning algorithms: while the overall output of a CCN is still invariant to the permutation of identical atoms, internally, the activations of the network are *covariant* rather than invariant, allowing us to better preserve information about the relationships between atoms.

3.8 GraphFlow Deep Learning Framework

3.8.1 Motivation for a new deep learning framework

Many Deep Learing frameworks have been proposed over the last decade. Among them, the most successful ones are TensorFlow (Abadi et al., 2016), PyTorch (Paszke et al., 2017), Mxnet (Chen et al., 2016), Theano (Al-Rfou et al., 2016). However, none of these frameworks are completely suitable for graph neural networks in the domain of molecular applications with high complexity tensor operations due to the following reasons:

1. The current frameworks are not flexible for an implementation of the **Virtual Indexing System** for efficient and low-cost tensor operations.
2. The most widely used Deep Learning framework - TensorFlow is incapable of constructing dynamic computation graphs during run time that is essential for graph neural networks which are dynamic in size and structure. To get rid of static computation graphs, Google Research has been proposed an extension of TensorFlow that

is TensorFlow-fold but has not completely solved the flexibility problem (Looks et al., 2017).

To address all these drawbacks, we implement from scratch our **GraphFlow Deep Learning Framework** (Hy and Jones, 2019) in C++11 with the following criteria:

1. Supports symbolic/automatic differentiation that allows users to construct any kind of neural networks without explicitly writing the complicated back-propagation code each time.
2. Supports dynamic computation graphs that is fundamental for graph neural networks such that partial computation graph is constructed before training and the rest is constructed during run time depending on the size and structure of the input graphs.
3. Supports sophisticated tensor/matrix operations with **Virtual Indexing System**.
4. Supports tensor/matrix operations implemented in CUDA for computation acceleration by GPUs.

3.8.2 Philosophy of the design

GraphFlow (Hy and Jones, 2019) is designed with the philosophy of Object Oriented Programming (OOP). There are several classes divided into the following groups:

1. **Data structures:** `Entity`, `Vector`, `Matrix`, `Tensor`, etc. Each of these components contain two arrays of floating-point numbers: `value` for storing the actual values, `gradient` for storing the gradients (that is the partial derivative of the loss function) for the purpose of automatic differentiation. Also, in each class, there are two functions: `forward()` and `backward()` in which `forward()` to evaluate the network values and `backward()` to compute the gradients. Based on the OOP philosophy, `Vector` inherits from `Entity`, and both `Matrix` and `Tensor` inherit from `Vector`, etc. It is essentially

important because polymorphism allows us to construct the computation graph of the neural network as a Directed Acyclic Graph (DAG) of `Entity` such that `forward()` and `backward()` functions of different classes can be called with object casting.

2. **Operators:** Matrix Multiplication, Tensor Contraction, Convolution, etc. For example, the matrix multiplication class `MatMul` inherits from `Matrix` class, and has 2 constructor parameters in `Matrix` type. Suppose that we have an object `A` of type `MatMul` that has 2 `Matrix` inputs `B` and `C`. In the `forward()` pass, `A` computes its value as `A = B * C` and stores it into `value` array. In the `backward()` pass, `A` got the gradients into `gradient` (as flowing from the loss function) and increase the gradients of `B` and `C`.

It is important to note that our computation graph is DAG and we find the topological order to evaluate `value` and `gradient` in the correct order. That means `A -> forward()` is called after both `B -> forward()` and `C -> forward()`, and `A -> backward()` is called before both `B -> backward()` and `C -> backward()`.

3. **Optimization algorithms:** Stochastic Gradient Descent (SGD), SGD with Momentum, Adam, AdaGrad, AdaMax, AdaDelta, etc. These algorithms are implemented into separate drivers: these drivers get the values and gradients of learnable parameters computed by the computation graph and then optimize the values of learnable parameters algorithmically.
4. **Neural Networks objects:** These are classes of neural network architectures implemented based on the core of GraphFlow including graph neural networks (for example, CCN, NGF and LCNN), convolutional neural networks, recurrent neural networks (for example, GRU and LSTM), multi-layer perceptron, etc. Each class has multiple supporting functions: load the trained learnable parameters from files, save them into files, learning with mini-batch or without mini-batch, using multi-threading or not, etc.

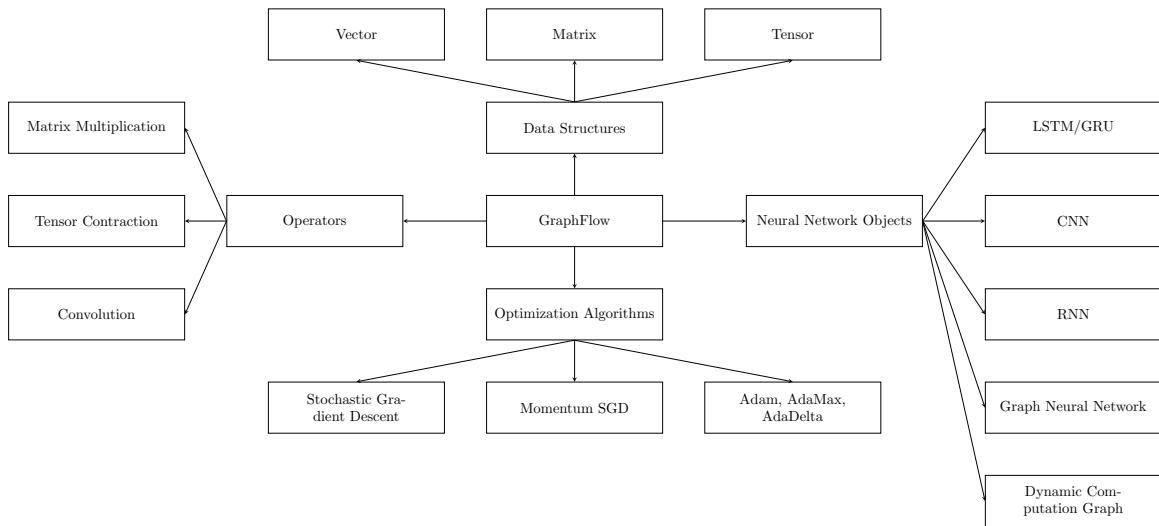


Figure 3.13: GraphFlow overview

Figure 3.13 describes the general structure of GraphFlow Deep Learning framework.

3.8.3 Parallelization

Efficient Matrix Multiplication In GPU

Multiple operations of a neural network can be expressed as matrix multiplication. Having a fast implementation of matrix multiplication is extremely important for a Deep Learning framework. We have implemented two versions of matrix multiplication in CUDA: one using naive kernel function that accesses matrices directly from the global memory of GPU, one is more sophisticated kernel function that uses shared memory in which the shared memory of each GPU block contains 2 blocks of the 2 input matrices to avoid latency of reading from the GPU global memory. Suppose that each GPU block can execute up to 512 threads concurrently, we select the block size as 22×22 . The second approach outperforms the first approach in our stress experiments. In general, it is important to algorithmically optimize our CUDA kernel to efficiently exploit the shared memory space, that would certainly give us a boost in the performance relatively comparing to the corresponding naive implementation

of using global memory space.

Efficient Tensor Contraction In CPU

Tensor stacking, tensor product, tensor contraction play the most important role in the success of Covariant Compositional Networks. Among them, tensor contraction is the most difficult operation to implement efficient due to the complexity of its algorithm. Let consider the second-order tensor product:

$$F_v^\ell \otimes A \downarrow_{\mathcal{P}_v^\ell}$$

where $F_v^\ell \in \mathbb{R}^{|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times c}$ is the result from tensor stacking operation of vertex v at level ℓ , $A \downarrow_{\mathcal{P}_v^\ell} \in \{0, 1\}^{|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell|}$ is the restricted adjacency matrix to the receptive field \mathcal{P}_v^ℓ , and c is the number of channels. With the Virtual Indexing System (see Fig. 3.14), we do not compute the full tensor product result, indeed we compute some elements of it when necessary. The task is to contract/reduce the tensor product $F_v^\ell \otimes A \downarrow_{\mathcal{P}_v^\ell}$ of 6th order into 3rd order tensor of size $|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times c$. For example, there are 18 ways of contractions in the second-order case. Suppose that our CPU has $N < 18$ cores, assuming that we can run all these cores concurrently, we launch N threads such that each thread processes $[18/N]$ contractions. There can be some threads doing more or less contractions. One challenge is about synchronization: we have to ensure that the updating operations are atomic ones.

Efficient Tensor Contraction In GPU

The real improvement in performance comes from GPU. Thus, in practice, we do not use the tensor contraction with multi-threading in CPU. Because we are experimenting on Tesla GPU K80, we have an assumption that each block of GPU can launch 512 threads and a GPU grid can execute 8 concurrent blocks. In GPU global memory, F_v^ℓ is stored as a float array of size $|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times c$, and the reduced adjacency matrix $A \downarrow_{\mathcal{P}_v^\ell}$ is stored as a float array of size $|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell|$. We divide the job to GPU in such a way that each thread processes

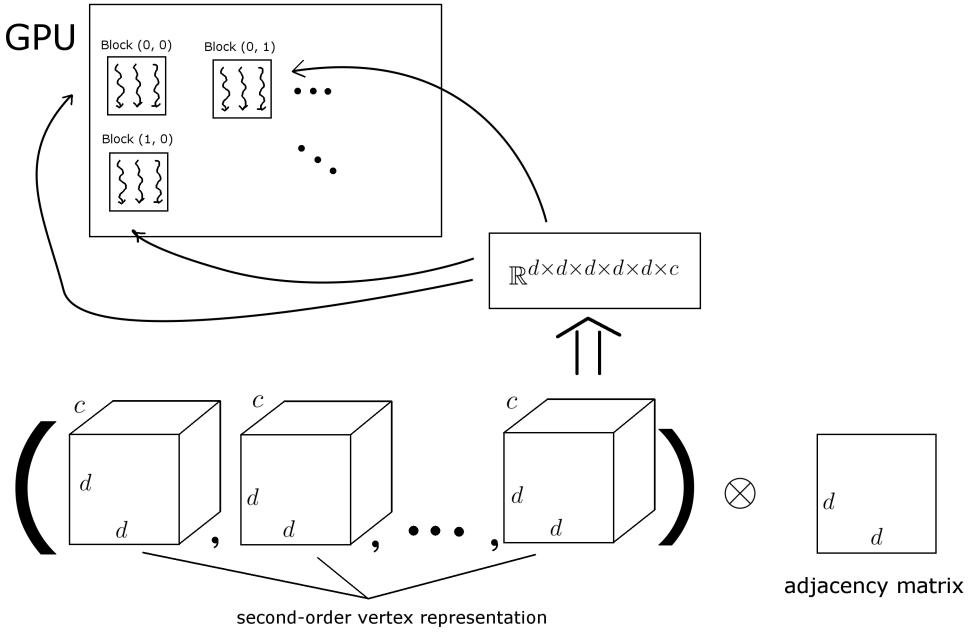


Figure 3.14: GPU implementations of tensor contractions in CCN 2D with the Virtual Indexing System. In this figure, d denotes the size of the graph and c denotes the number of channels.

a part of F_v^ℓ and a part of $A \downarrow_{\mathcal{P}_v^\ell}$. We assign the computation work equally among threads based on the estimated asymptotic complexity.

Again, synchronization is also a real challenge: all the updating operations must be the atomic ones. However, having too many atomic operations can slow down our concurrent algorithm. That is why we have to design our GPU algorithm with the minimum number of atomic operations as possible. We obtain a much better performance with GPU after careful consideration of all factors.

CPU Multi-threading In Gradient Computation

Given a minibatch of M training examples, it is a natural idea that we can separate the gradient computation jobs into multiple threads such that each thread processes exactly one

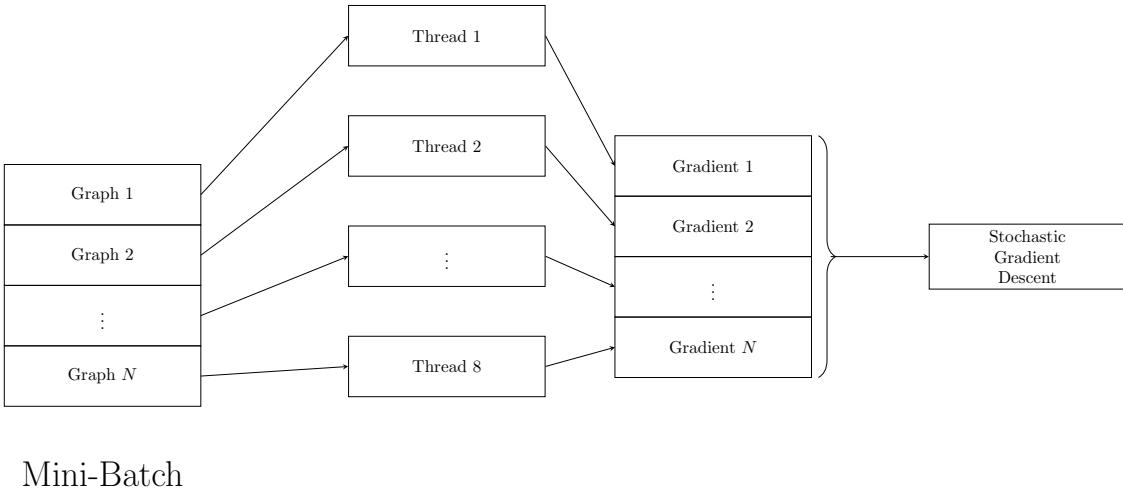


Figure 3.15: CPU multi-threading for gradient computation

training example at a time before continuing to process the next example. We have to make sure that there is no overlapping among these threads. After completing the gradient computations from all these M training examples, we sum up all the gradients, average them by M and apply an variant of Stochastic Gradient Descent to optimize the neural networks before moving to the next minibatch. Technically, suppose that we can execute T threads concurrent at a time for gradient computation jobs. Before every training starts, we initialize exactly T identical dynamic computation graphs by GraphFlow. Given a minibatch of M training examples, we distribute the examples to T thread, each thread uses a different dynamic computation graph for its gradient computation job. By this way, there is absolutely no overlapping and our training is completely synchronous. The minibatch training with CPU multi-threading is described by figure 3.15.

Reducing data movement between GPU and main memory

One challenge of using GPU is that we must move the data from the main memory to the GPU before performing any computation. This could prevent us from achieving high effi-

cient computation since data movement takes time and the GPU cannot be used until this process completes. We solve this problem by detaching data movement from computation. We introduce two new functions, `upload` and `download`, let them handle the data movement to and from the GPU, respectively. The `forward` and `backward` functions only perform computation. This approach makes the framework more flexible as it can dynamically determine which parts of the data should be moved and when to move them. Therefore, the framework has more options when scheduling computation flows of the network, enabling better GPU utilization as well as avoiding unnecessary communication caused by poor implementations. Figure 3.16 shows an example of our approach. On the left of the figure is the C++ code for computing matrix computation of $A \times B \times C$. The execution of the code is depicted on the right side. Firstly, the framework copies necessary data from main memory to GPU's global memory by calling `upload` function. The result and gradient is then generated after the calls to `forward()` and `backward()` functions. Those functions work in the way similar to what we introduced in previous sections except that the computation is performed entirely by the GPU. In the end, `download` function is called to move the computation results back to the main memory. Clearly, data movement occurs only during the initialization and finalization of the whole process, there is no communication between GPU and main memory during the computation so the performance would be improved significantly.

3.8.4 Performance evaluation

In this section, we evaluate the computing performance of GraphFlow Deep Learning Framework in multiple settings.

On tensor contraction

We also compare the GPU implementation of tensor contraction with the CPU one. We want to remark that the tensor contraction complexity is $O(18 \times |\mathcal{P}_v^\ell|^5 \times c)$ that grows expo-

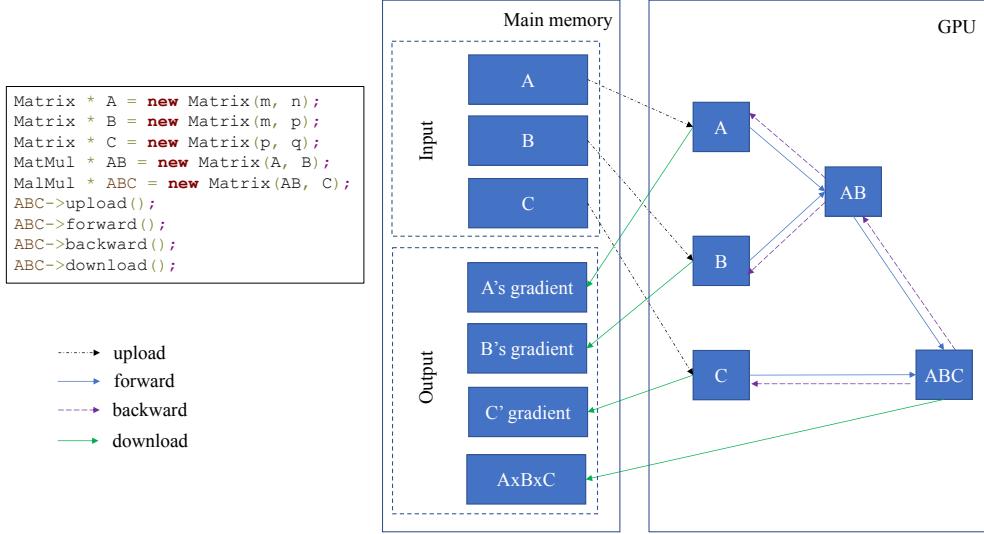


Figure 3.16: Example of data flow between GPU and main memory

	$ \mathcal{P}_v^\ell $	c	Floating-points	CPU	GPU
5	10		562,500	3 ms	3 ms
5	20		1,125,000	7 ms	1 ms
10	10		18,000,000	56 ms	1 ms
10	20		36,000,000	103 ms	3 ms
20	10		576,000,000	977 ms	18 ms
20	20		1,152,000,000	2,048 ms	27 ms
35	10		9,453,937,500	12,153 ms	267 ms
35	20		18,907,875,000	25,949 ms	419 ms

Table 3.1: GPU vs CPU tensor contraction running time (milliseconds)

nentially with the size of receptive field $|\mathcal{P}_v^\ell|$ and grows linearly with the number of channels c . We have a constant 18 as the number of contractions implemented in the second-order case. We have several tests with the size of receptive field $|\mathcal{P}_v^\ell|$ ranging in $\{5, 10, 20, 35\}$ and the number of channels c ranging in $\{10, 20\}$. In the largest case with $|\mathcal{P}_v^\ell| = 35$ and $c = 20$, we observe that GPU gives a factor of approximately 62x speedup. Table 3.1 and figure 3.17 show the details. Figure 3.14 describes the general idea of GPU implementation of tensor contractions in CCN 2D by *Virtual Indexing System*.

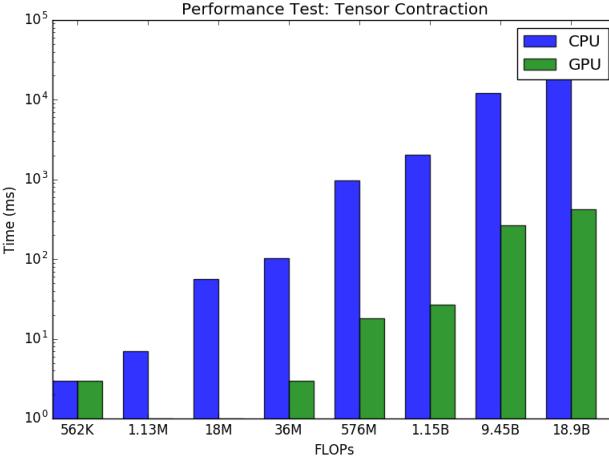


Figure 3.17: GPU vs CPU tensor contraction running time (milliseconds) in \log_{10} scale

$ V $	Max $ \mathcal{P}_v^\ell $	c	L	CPU	GPU
10	10	10	6	1,560 ms	567 ms
15	10	10	6	1,664 ms	543 ms
20	15	10	6	7,684 ms	1,529 ms
25	15	10	6	11,777 ms	1,939 ms

Table 3.2: GPU and CPU network evaluation running time (milliseconds)

On synthetic random graphs

In this experiment, we generate synthetic random input graphs by Erdos-Renyi $p = 0.5$ model. The number of vertices $|V| \in \{10, 15, 20, 25\}$. We fix the maximum size of receptive field $|\mathcal{P}_v^\ell|$ as 10 and 15, the number of channels c as 10, and the number of levels/layers of the neural network L as 6. In the largest case of the graph with 25 vertices, GPU gives a factor of approximately 6x speedup. Table 3.2 shows the detail.

On small molecular dataset

This is the total training and testing time on a small dataset of 4 molecules CH_4 , NH_3 , H_2O , C_2H_4 with 1,024 epochs. After each epoch, we evaluate the neural network immediately. CCN 1D denotes the Covariant Compositional Networks with the first-order representation, the number of layers/levels is in $\{1, 2, 4, 8, 16\}$. CCN 2D denotes the Covariant Composi-

Model	Layers	Single-thread	Multi-thread
CCN 1D	1	1,836 ms	874 ms
CCN 1D	2	4,142 ms	1,656 ms
CCN 1D	4	9,574 ms	3,662 ms
CCN 1D	8 (deep)	20,581 ms	7,628 ms
CCN 1D	16 (very deep)	42,532 ms	15,741 ms
CCN 2D	1	35 seconds	10 seconds
CCN 2D	2	161 seconds	49 seconds

Table 3.3: Single thread vs Multiple threads running time

tional Networks with the second-order representation, the number of layers/levels is in {1, 2}. The number of channels $c = 10$ in all settings. In this experiment, we use 4 threads for the training minibatches of 4 molecules and compare the running time with the single thread case. All models are fully converged. Table 3.3 shows the details.

3.9 Experiments

We now compare our CCN framework to several standard graph learning algorithms. We focus on two datasets that contain the result of a large number of Density Functional Theory (DFT) calculations:

1. **The Harvard Clean Energy Project (HCEP)**, consisting of 2.3 million organic compounds that are candidates for use in solar cells (Hachmann et al., 2011).
2. **QM9**, a dataset of ~134k organic molecules with up to nine heavy atoms (C, O, N and F) (Ramakrishnan et al., 2014) out of the GDB-17 universe of molecules (Ruddigkeit et al., 2012). Each molecule contains data including 13 target chemical properties, along with the spatial position of every constituent atom.

We are interested in the ability of our algorithm to learn on both pure graphs, and also on physical data. As such, we perform three experiments. We start with two experiments based

only upon atomic identity and molecular graph topology:

1. **HCEP**: We use a random sample of 50,000 molecules of the HCEP dataset; our learning target is Power Conversion Efficiency (PCE), and we present the mean average error (MAE). The input vertex feature l_v is a one-hot vector of atomic identity concatenated with purely synthesized graph-based features.
2. **QM9(a)**: We predict the 13 target properties of every molecule. For this text we consider only heavy atoms and exclude hydrogen. Vertex feature initialization is performed in the same manner as the HCEP experiment. For training the neural networks, we normalized all 13 learning targets to have mean 0 and standard deviation 1. We report the MAE with respect to the normalized learning targets.

We also tested our algorithm's ability to learn on DFT data based upon physical features. We perform the following experiment:

3. **QM9(b)**: The QM9 dataset with each molecule including hydrogen atoms. We use both physical atomic information (vertex features) and bond information (edge features) including: atom type, atomic number, acceptor, donor, aromatic, hybridization, number of hydrogens, Euclidean distance and Coulomb distance between pair of atoms. All the information is encoded in a vectorized format.

To include the edge features into our model along with the vertex features, we used the concept of a *line graph* from graph theory. We constructed the line graph for each molecular graph in such a way that: an edge of the molecular graph corresponds to a vertex in its line graph, and if two edges in the molecular graph share a common vertex then there is an edge between the two corresponding vertices in the line graph. (See Fig. 3.22). The edge features become vertex features in the line graph. The inputs of our model contain both the molecular graph and its line graph. The feature vectors

F_ℓ between the two graphs are merged at each level ℓ . (See step 12 of the algorithm 15).

In QM9(b), we report the mean average error for each learning target in its corresponding physical unit and compare it against the Density Functional Theory (DFT) error given by (Faber et al., 2017).

In the case of HCEP, we compared CCNs to lasso, ridge regression, random forests, gradient boosted trees, optimal assignment Weisfeiler–Lehman graph kernel (Kriege et al., 2016) (WL), neural graph fingerprints (Duvenaud et al., 2015), and the “patchy-SAN” convolutional type algorithm (referred to as PSCN) (Niepert et al., 2016). For the first four of these baseline methods, we created simple feature vectors from each molecule: the number of bonds of each type (i.e., number of H–H bonds, number of C–O bonds, etc.) and the number of atoms of each type. Molecular graph fingerprints uses atom labels of each vertex as base features. For ridge regression and lasso, we cross validated over λ . For random forests and gradient boosted trees, we used 400 trees, and cross validated over max depth, minimum samples for a leaf, minimum samples to split a node, and learning rate (for GBT). For neural graph fingerprints, we used 3 layers and a hidden layer size of 10. In PSCN, we used a patch size of 10 with two convolutional layers and a dense layer on top as described in their paper.

For QM9(a), we compared against the Weisfeiler–Lehman graph kernel, neural graph fingerprints, and PSCN. The settings for NGF and PSCN are as described for HCEP. For QM9(b), we compared against DFT error provided in (Faber et al., 2017).

We initialized the synthesized graph-based features of each vertex with computed histogram alignment features, inspired by (Kriege et al., 2016), of depth up to 10. Each vertex receives a base label $l_v = \text{concat}_{d=1}^{10} H_v^d$ where $H_v^d \in \mathbb{R}^c$ (with c being the total number of distinct

discrete node labels) is the vector of relative frequencies of each label for the set of vertices at distance equal to d from vertex v . Our CCNs architecture contains up to five levels.

In each experiment we separated 80% of the dataset for training, 10% for validation, and evaluated on the remaining 10% test set. We used Adam optimization (Kingma and Ba, 2015) with the initial learning rate set to 0.001 after experimenting on a held out validation set. The learning rate decayed linearly after each step towards a minimum of 10^{-6} .

Our method, Covariant Compositional Networks, and other graph neural networks such as Neural Graph Fingerprints (Duvenaud et al., 2015), PSCN (Niepert et al., 2016) and Gated Graph Neural Networks (Li et al., 2016a) are implemented based on the GraphFlow framework (see section 3.8).

Tables 3.4, 3.5, and 3.6 show the results of HCEP, QM9(a) and QM9(b) experiments, respectively. Figures 3.18 and 3.19 show the 2D PCA projections of *learned* molecular representations in HCEP dataset with Weisfeiler-Lehman, Covariant Compositional Networks 1D & 2D, respectively. On the another hand, figures 3.20 and 3.21 show the 2D projections with t-SNE (Maaten and Hinton, 2008). The colors represent the PCE values ranging from 0 to 11. Figure 3.23 shows the distributions between ground-truth and prediction of CCN 1D & 2D in HCEP.

On the subsampled HCEP dataset, CCN outperforms all other methods by a very large margin. In the QM9(a) experiment, CCN obtains better results than three other graph learning algorithms for all 13 learning targets. In the QM9(b) experiment, our method gets smaller errors comparing to the DFT calculation in 11 out of 12 learning targets (we do not have the DFT error for R2).

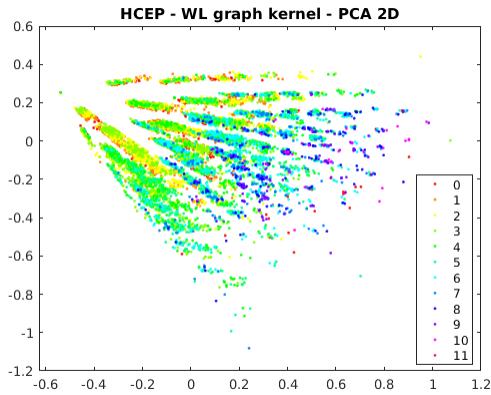


Figure 3.18: 2D PCA projections of Weisfeiler-Lehman features in HCEP

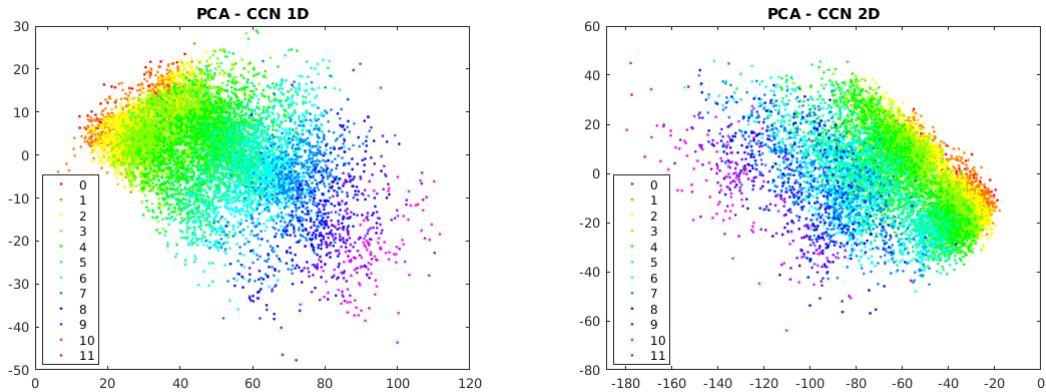


Figure 3.19: 2D PCA projections of CCNs graph representations in HCEP

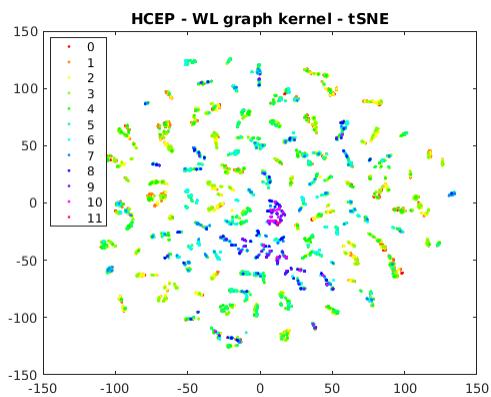


Figure 3.20: 2D t-SNE projections of Weisfeiler-Lehman features in HCEP

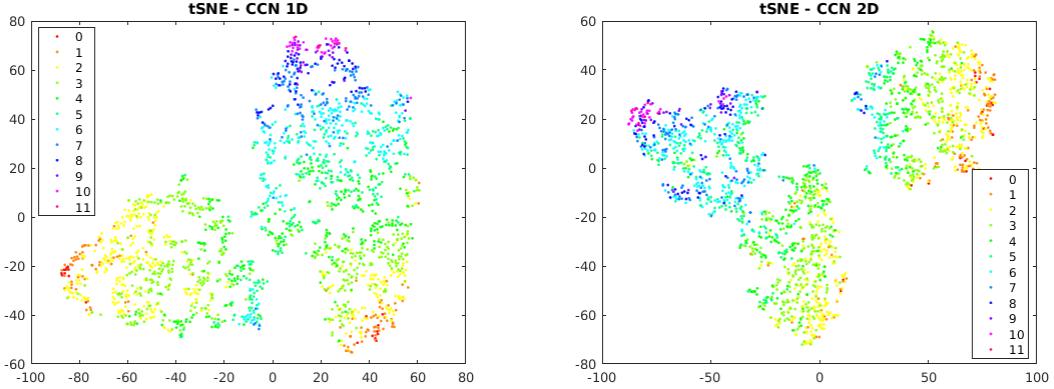


Figure 3.21: 2D t-SNE projections of CCNs graph representations in HCEP

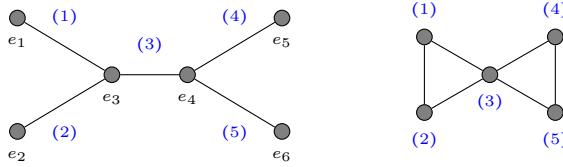


Figure 3.22: Molecular graph of C_2H_4 (left) and its corresponding line graph (right).

3.10 Software

The source code of GraphFlow including the implementation of Covariant Compositional Networks (CCNs) and various forms of graph neural networks can be found at

<https://github.com/HyTruongSon/GraphFlow>.

In addition, our easy-to-use and efficient implementation of CCNs with TensorFlow and PyTorch's APIs based on a shared common C++ core can be found at

<https://github.com/HyTruongSon/LibCCNs>.

Finally, our PyTorch implementation of a special case of our general architecture, that is the Invariant and Equivariant Graph Networks introduced by (Maron et al., 2019b), is publicly available at

<https://github.com/HyTruongSon/InvariantGraphNetworks-PyTorch>.

	Test MAE	Test RMSE
Lasso	0.867	1.437
Ridge regression	0.854	1.376
Random forest	1.004	1.799
Gradient boosted trees	0.704	1.005
WL graph kernel	0.805	1.096
Neural graph fingerprints	0.851	1.177
PSCN	0.718	0.973
CCN 1D	0.216	0.291
CCN 2D	0.340	0.449

Table 3.4: HCEP regression results

Target	WLGK	NGF	PSCN	CCN 2D
alpha	0.46	0.43	0.20	0.16
Cv	0.59	0.47	0.27	0.23
G	0.51	0.46	0.33	0.29
gap	0.72	0.67	0.60	0.54
H	0.52	0.47	0.34	0.30
HOMO	0.64	0.58	0.51	0.39
LUMO	0.70	0.65	0.59	0.53
mu	0.69	0.63	0.54	0.48
omega1	0.72	0.63	0.57	0.45
R2	0.55	0.49	0.22	0.19
U	0.52	0.47	0.34	0.29
U0	0.52	0.47	0.34	0.29
ZPVE	0.57	0.51	0.43	0.39

Table 3.5: QM9(a) regression results (MAE)

3.11 Chapter Conclusion

In this chapter, we presented a general framework called covariant compositional networks (CCNs) for learning the properties of molecules from their graphs. Central to this framework are two key ideas: (1) a compositional structure that generalizes message passing neural networks (MPNNs) and (2) the concept of covariant aggregation functions based on tensor algebra. These tensor aggregation rules ensure the network to process graphs in a permutation-equivariant way for each layer of the (higher-order) message passing scheme,

Target	CCNs	DFT error	Physical unit
alpha	0.19	0.4	Bohr ³
Cv	0.06	0.34	cal/mol/K
G	0.05	0.1	eV
gap	0.11	1.2	eV
H	0.05	0.1	eV
HOMO	0.08	2.0	eV
LUMO	0.07	2.6	eV
mu	0.43	0.1	Debye
omegal	2.54	28	cm ⁻¹
R2	5.03	-	Bohr ²
U	0.06	0.1	eV
U0	0.05	0.1	eV
ZPVE	0.0043	0.0097	eV

Table 3.6: QM9(b) regression results (MAE)

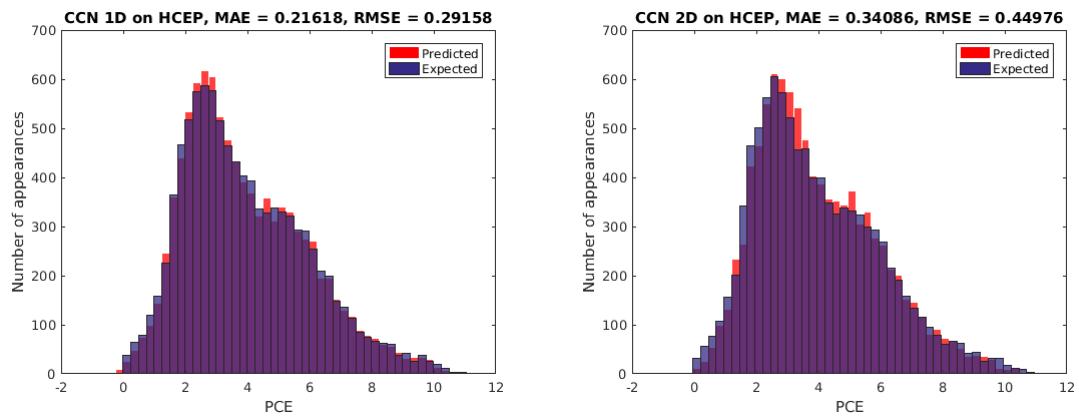


Figure 3.23: Distributions of ground-truth and prediction of CCN 1D & 2D in HCEP

and return a permutation-invariant output from the top of the network.

We argue that CCNs can extract multiscale structure from molecular graphs and keep track of the local topology in a manner the conventional MPNNs are not able to. We also introduced the GraphFlow software library that provides an efficient implementation of CCNs and various forms of graph neural networks in C++/CUDA. By using our new deep learning framework GraphFlow, we were able to show that CCNs often outperform existing state-of-the-art algorithms in learning molecular properties. This represents a significant advancement in data-driven quantum chemistry, and enables the exploration of large sets of molecules.

CHAPTER 4

MULTIRESOLUTION EQUIVARIANT GRAPH VARIATIONAL AUTOENCODER

4.1 Chapter Introduction

Understanding graphs in a multiscale and multiresolution perspective is essential for capturing the structure of molecules, social networks, or the World Wide Web. Graph neural networks (GNNs) utilizing various ways of generalizing the concept of convolution to graphs (Scarselli et al., 2009) (Niepert et al., 2016) (Li et al., 2016b) have been widely applied to many learning tasks, including modeling physical systems (Battaglia et al., 2016), finding molecular representations to estimate quantum chemical computation (Duvenaud et al., 2015) (Kearnes et al., 2016) (Gilmer et al., 2017) (Hy et al., 2018), and protein interface prediction (Fout et al., 2017). One of the most popular types of GNNs is message passing neural nets (MPNNs) that are constructed based on the message passing scheme in which each node propagates and aggregates information, encoded by vectorized messages, to and from its local neighborhood. While this framework has been immensely successful in many applications, it lacks the ability to capture the multiscale and multiresolution structures that are present in complex graphs (Rustamov and Guibas, 2013) (Chen et al., 2014) (Cheng et al., 2015) (Xu et al., 2019).

Ying et al. (2018) proposed a multiresolution graph neural network that employs a differential pooling operator to coarsen the graph. While this approach is effective in some settings, it is based on *soft* assignment matrices, which means that (a) the sparsity of the graph is quickly lost in higher layers (b) the algorithm isn't able to learn an actual hard clustering of the vertices. The latter is important in applications such as learning molecular graphs, where clusters should ideally be interpretable as concrete subunits of the graphs,

e.g., functional groups.

In contrast, in this paper we propose an arhitecture called *Multiresolution Graph Network* (*MGN*), and its generative cousin, *Multiresolution Graph Variational Autoencoder* (*MGVAE*), which explicitly learn a multilevel hard clustering of the vertices, leading to a true multiresolution hierarchy. In the decoding stage, to “uncoarsen” the graph, MGVAE needs to generate local adjacency matrices, which is inherently a second order task with respect to the action of permutations on the vertices, hence MGVAE needs to leverage the recently developed framework of higher order permutation equivariant message passing networks (Hy et al., 2018; Maron et al., 2019b). MGN allows us to extend the existing model of variational autoencoders (VAEs) with a hierarchy of latent distributions that can stochastically generate a graph in multiple resolution levels. Our experiments show that having a flexible clustering procedure from MGN enables MGVAE to detect, reconstruct and finally generate important graph substructures, especially chemical functional groups.

4.2 Related work

There have been significant advances in understanding the invariance and equivariance properties of neural networks in general (Cohen and Welling, 2016) (Cohen and Welling, 2017), of graph neural networks (Maron et al., 2019b), of neural networks learning on sets (Zaheer et al., 2017) (Serviansky et al., 2020) (Maron et al., 2020), along with their expressive power on graphs (Maron et al., 2019c) (Maron et al., 2019a). Our work is in line with group equivariant networks operating on graphs and sets. Multiscale, multilevel, multiresolution and coarse-grained techniques have been widely applied to graphs and discrete domains such as diffusion wavelets (Coifman and Maggioni, 2006); spectral wavelets on graphs (Hammond et al., 2011); finding graph wavelets based on partitioning/clustering (Rustamov and Guibas, 2013); graph clustering and finding balanced cuts on large graphs (Dhillon et al.,

2005) (Dhillon et al., 2007) (Chiang et al., 2012) (Si et al., 2014); and link prediction on social networks (Shin et al., 2012). Prior to our work, some authors such as (Zhou et al., 2019) proposed a multiscale generative model on graphs using GAN (Goodfellow et al., 2014), but the hierarchical structure was built by heuristics algorithm, not learnable and not flexible to new data that is also an existing limitation of the field. In general, our work exploits the powerful group equivariant networks to encode a graph and to learn to form balanced partitions via back-propagation in a data-driven manner without using any heuristics as in the existing works.

In the field of deep generative models, it is generally recognized that introducing a hierarchy of latents and adding stochasticity among latents leads to more powerful models capable of learning more complicated distributions (Blei et al., 2003) (Ranganath et al., 2016) (Ingramham and Marks, 2017) (Klushyn et al., 2019) (Wu et al., 2020) (Vahdat and Kautz, 2020). Our work combines the hierarchical variational autoencoder with learning to construct the hierarchy that results into a generative model able to generate graphs at many resolution levels.

4.3 Multiresolution graph network

4.3.1 Construction

An undirected weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{F}_v, \mathcal{F}_e)$ with node set \mathcal{V} and edge set \mathcal{E} is represented by an adjacency matrix $\mathcal{A} \in \mathbb{N}^{|\mathcal{V}| \times |\mathcal{V}|}$, where $\mathcal{A}_{ij} > 0$ implies an edge between node v_i and v_j with weight \mathcal{A}_{ij} (e.g., $\mathcal{A}_{ij} \in \{0, 1\}$ in the case of unweighted graph); while node features are represented by a matrix $\mathcal{F}_v \in \mathbb{R}^{|\mathcal{V}| \times d_v}$, and edge features are represented by a tensor $\mathcal{F}_e \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}| \times d_e}$. The second-order tensor representation of edge features is necessary for our higher-order message passing networks described in the next section. Indeed, \mathcal{F}_v can

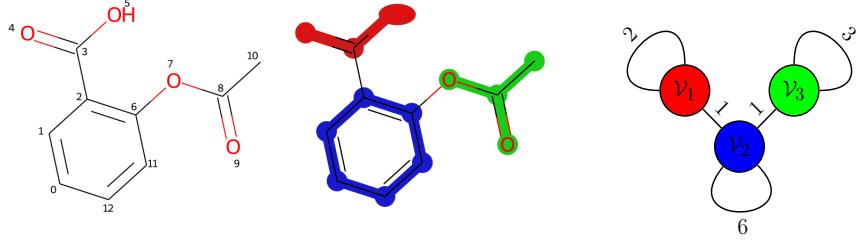


Figure 4.1: Aspirin $C_9H_8O_4$, its 3-cluster partition and the corresponding coarsen graph

be encoded in the diagonal of \mathcal{F}_e .

Definition 4.3.1. A K -cluster partition of graph \mathcal{G} is a partition of the set of nodes \mathcal{V} into K mutually exclusive clusters $\mathcal{V}_1, \dots, \mathcal{V}_K$. Each cluster corresponds to an induced subgraph $\mathcal{G}_k = \mathcal{G}[\mathcal{V}_k]$.

Definition 4.3.2. A coarsening of \mathcal{G} is a graph $\tilde{\mathcal{G}}$ of K nodes defined by a K -cluster partition in which node \tilde{v}_k of $\tilde{\mathcal{G}}$ corresponds to the induced subgraph \mathcal{G}_k . The weighted adjacency matrix $\tilde{\mathcal{A}} \in \mathbb{N}^{K \times K}$ of $\tilde{\mathcal{G}}$ is

$$\tilde{\mathcal{A}}_{kk'} = \begin{cases} \frac{1}{2} \sum_{v_i, v_j \in \mathcal{V}_k} \mathcal{A}_{ij}, & \text{if } k = k', \\ \sum_{v_i \in \mathcal{V}_k, v_j \in \mathcal{V}_{k'}} \mathcal{A}_{ij}, & \text{if } k \neq k', \end{cases}$$

where the diagonal of $\tilde{\mathcal{A}}$ denotes the number of edges inside each cluster, while the off-diagonal denotes the number of edges between two clusters.

Fig. 4.3.1 shows an example of Defs. 4.3.1 and 4.3.2: a 3-cluster partition of the Aspirin $C_9H_8O_4$ molecular graph and its coarsening graph. Def. 4.3.3 defines the multiresolution of graph \mathcal{G} in a bottom-up manner in which the bottom level is the highest resolution (e.g., \mathcal{G} itself) while the top level is the lowest resolution (e.g., \mathcal{G} is coarsened into a single node).

Definition 4.3.3. An L -level coarsening of a graph \mathcal{G} is a series of L graphs $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(L)}$ in which

1. $\mathcal{G}^{(L)}$ is \mathcal{G} itself.
2. For $1 \leq \ell \leq L - 1$, $\mathcal{G}^{(\ell)}$ is a coarsening graph of $\mathcal{G}^{(\ell+1)}$ as defined in Def. 4.3.2. The number of nodes in $\mathcal{G}^{(\ell)}$ is equal to the number of clusters in $\mathcal{G}^{(\ell+1)}$.
3. The top level coarsening $\mathcal{G}^{(1)}$ is a graph consisting of a single node, and the corresponding adjacency matrix $\mathcal{A}^{(1)} \in \mathbb{N}^{1 \times 1}$.

Definition 4.3.4. An L -level Multiresolution Graph Network (MGN) of a graph \mathcal{G} consists of $L - 1$ tuples of five network components $\{(\mathbf{c}^{(\ell)}, \mathbf{e}_{\text{local}}^{(\ell)}, \mathbf{d}_{\text{local}}^{(\ell)}, \mathbf{d}_{\text{global}}^{(\ell)}, \mathbf{p}^{(\ell)})\}_{\ell=2}^L$. The ℓ -th tuple encodes $\mathcal{G}^{(\ell)}$ and transforms it into a lower resolution graph $\mathcal{G}^{(\ell-1)}$ in the higher level. Each of these network components has a separate set of learnable parameters $(\boldsymbol{\theta}_1^{(\ell)}, \boldsymbol{\theta}_2^{(\ell)}, \boldsymbol{\theta}_3^{(\ell)}, \boldsymbol{\theta}_4^{(\ell)}, \boldsymbol{\theta}_5^{(\ell)})$. For simplicity, we collectively denote the learnable parameters as $\boldsymbol{\theta}$, and drop the superscript. The network components are defined as follows:

1. Clustering procedure $\mathbf{c}(\mathcal{G}^{(\ell)}; \boldsymbol{\theta})$, which partitions graph $\mathcal{G}^{(\ell)}$ into K clusters $\mathcal{V}_1^{(\ell)}, \dots, \mathcal{V}_K^{(\ell)}$. Each cluster is an induced subgraph $\mathcal{G}_k^{(\ell)}$ of $\mathcal{G}^{(\ell)}$ with adjacency matrix $\mathcal{A}_k^{(\ell)}$.
2. Local encoder $\mathbf{e}_{\text{local}}(\mathcal{G}_k^{(\ell)}; \boldsymbol{\theta})$, which is a permutation equivariant (see Defs. 4.3.5, 4.3.6) graph neural network that takes as input the subgraph $\mathcal{G}_k^{(\ell)}$, and outputs a set of node latents $\mathcal{Z}_k^{(\ell)}$ represented as a matrix of size $|\mathcal{V}_k^{(\ell)}| \times d_z$.
3. Local decoder $\mathbf{d}_{\text{local}}(\mathcal{Z}_k^{(\ell)}; \boldsymbol{\theta})$, which is a permutation equivariant neural network that tries to reconstruct the subgraph adjacency matrix $\mathcal{A}_k^{(\ell)}$ for each cluster from the local encoder's output latents.
4. (Optional) Global decoder $\mathbf{d}_{\text{global}}(\mathcal{Z}^{(\ell)}; \boldsymbol{\theta})$, which is a permutation equivariant neural network that reconstructs the full adjacency matrix $\mathcal{A}^{(\ell)}$ from all the node latents of K clusters $\mathcal{Z}^{(\ell)} = \bigoplus_k \mathcal{Z}_k^{(\ell)}$ represented as a matrix of size $|\mathcal{V}^{(\ell)}| \times d_z$.
5. Pooling network $\mathbf{p}(\mathcal{Z}_k^{(\ell)}; \boldsymbol{\theta})$, which is a permutation invariant (see Defs. 4.3.5, 4.3.6) neural network that takes the set of node latents $\mathcal{Z}_k^{(\ell)}$ and outputs a single cluster

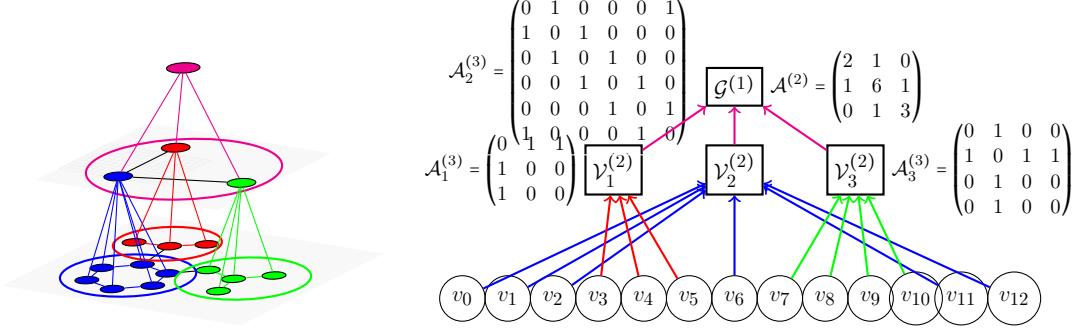


Figure 4.2: Hierarchy of 3-level Multiresolution Graph Network on Aspirin molecular graph

latent $\tilde{z}_k^{(\ell)} \in d_z$. The coarsening graph $\mathcal{G}^{(\ell-1)}$ has adjacency matrix $\mathcal{A}^{(\ell-1)}$ built as in Def. 4.3.2, and the corresponding node features $\mathcal{Z}^{(\ell-1)} = \bigoplus_k \tilde{z}_k^{(\ell)}$ represented as a matrix of size $K \times d_z$.

Algorithmically, MGN works in a bottom-up manner as a tree-like hierarchy starting from the highest resolution graph $\mathcal{G}^{(L)}$, going to the lowest resolution $\mathcal{G}^{(1)}$ (see Fig. 4.3.1). Iteratively, at ℓ -th level, MGN partitions the current graph into K clusters by running the clustering procedure $\mathbf{c}^{(\ell)}$. Then, the local encoder $\mathbf{e}_{\text{local}}^{(\ell)}$ and local decoder $\mathbf{d}_{\text{global}}^{(\ell)}$ operate on each of the K subgraphs separately, and can be executed in parallel. This encoder/decoder pair is responsible for capturing the local structures. Finally, the pooling network $\mathbf{p}^{(\ell)}$ shrinks each cluster into a single node of the next level. Optionally, the global decoder $\mathbf{d}_{\text{global}}^{(\ell)}$ makes sure that the whole set of node latents $\mathcal{Z}^{(\ell)}$ is able to capture the inter-connection between clusters.

In terms of time and space complexity, MGN is more efficient than existing methods in the field. The cost of global decoding the highest resolution graph is proportional to $|\mathcal{V}|^2$. For example, while the encoder can exploit the sparsity of the graph and has complexity $\mathcal{O}(|\mathcal{E}|)$, a simple dot-product decoder $\mathbf{d}_{\text{global}}(\mathcal{Z}) = \text{sigmoid}(\mathcal{Z}\mathcal{Z}^T)$ has both time and space complexity of $\mathcal{O}(|\mathcal{V}|^2)$ which is infeasible for large graphs. In contrast, the cost of running

K local dot-product decoders is $\mathcal{O}(|\mathcal{V}|^2/K)$, which is approximately K times more efficient.

4.3.2 Higher order message passing

In this paper we consider permutation symmetry, i.e., symmetry to the action of the symmetric group, \mathbb{S}_n . An element $\sigma \in \mathbb{S}_n$ is a permutation of order n , or a bijective map from $\{1, \dots, n\}$ to $\{1, \dots, n\}$. The action of \mathbb{S}_n on an adjacency matrix $\mathcal{A} \in \mathbb{R}^{n \times n}$ and on a latent matrix $\mathcal{Z} \in \mathbb{R}^{n \times d_z}$ are

$$[\sigma \cdot \mathcal{A}]_{i_1, i_2} = \mathcal{A}_{\sigma^{-1}(i_1), \sigma^{-1}(i_2)}, \quad [\sigma \cdot \mathcal{Z}]_{i, j} = \mathcal{Z}_{\sigma^{-1}(i), j}, \quad \sigma \in \mathbb{S}_n.$$

Here, the adjacency matrix \mathcal{A} is a second order tensor with a single feature channel, while the latent matrix \mathcal{Z} is a first order tensor with d_z feature channels. In general, the action of \mathbb{S}_n on a k -th order tensor $\mathcal{X} \in \mathbb{R}^{n^k \times d}$ (the last index denotes the feature channels) is defined similarly as:

$$[\sigma \cdot \mathcal{X}]_{i_1, \dots, i_k, j} = \mathcal{X}_{\sigma^{-1}(i_1), \dots, \sigma^{-1}(i_k), j}, \quad \sigma \in \mathbb{S}_n.$$

Network components of MGN (as defined in Sec. 4.3.1) at each resolution level must be either *equivariant*, or *invariant* with respect to the permutation action on the node order of $\mathcal{G}^{(\ell)}$. Formally, we define these properties in Def. 4.3.5.

Definition 4.3.5. An \mathbb{S}_n -equivariant (or permutation equivariant) function is a function $f: \mathbb{R}^{n^k \times d} \rightarrow \mathbb{R}^{n^{k'} \times d'}$ that satisfies $f(\sigma \cdot \mathcal{X}) = \sigma \cdot f(\mathcal{X})$ for all $\sigma \in \mathbb{S}_n$ and $\mathcal{X} \in \mathbb{R}^{n^k \times d}$. Similarly, we say that f is \mathbb{S}_n -invariant (or permutation invariant) if and only if $f(\sigma \cdot \mathcal{X}) = f(\mathcal{X})$.

Definition 4.3.6. An \mathbb{S}_n -equivariant network is a function $f: \mathbb{R}^{n^k \times d} \rightarrow \mathbb{R}^{n^{k'} \times d'}$ defined as a composition of \mathbb{S}_n -equivariant linear functions f_1, \dots, f_T and \mathbb{S}_n -equivariant nonlinearities $\gamma_1, \dots, \gamma_T$:

$$f = \gamma_T \circ f_T \circ \dots \circ \gamma_1 \circ f_1.$$

On the other hand, an \mathbb{S}_n -invariant network is a function $f : \mathbb{R}^{n^k \times d} \rightarrow \mathbb{R}$ defined as a composition of an \mathbb{S}_n -equivariant network f' and an \mathbb{S}_n -invariant function on top of it, e.g., $f = f'' \circ f'$.

In order to build higher order equivariant networks, we revisit some basic tensor operations: tensor product (see Def. 4.3.7) and tensor contraction (see Def. 4.3.8). It can be shown that these tensor operations respect permutation equivariance (Kondor et al., 2018a) (Hy et al., 2018). Based on them, we build our second order message passing networks.

Definition 4.3.7. The **tensor product** of $A \in \mathbb{R}^{n^a}$ with $B \in \mathbb{R}^{n^b}$ yields a tensor $C = A \otimes B \in \mathbb{R}^{n^{a+b}}$ where

$$C_{i_1, i_2, \dots, i_{a+b}} = A_{i_1, i_2, \dots, i_a} B_{i_{a+1}, i_{a+2}, \dots, i_{a+b}}$$

Definition 4.3.8. The **contraction** of $A \in \mathbb{R}^{n^a}$ along the pair of dimensions $\{x, y\}$ (assuming $x < y$) yields a $(a - 2)$ -th order tensor

$$C_{i_1, \dots, i_{x-1}, j, i_{x+1}, \dots, i_{y-1}, j, i_{y+1}, \dots, i_a} = \sum_{i_x, i_y} A_{i_1, \dots, i_a}$$

where we assume that i_x and i_y have been removed from amongst the indices of C . Using Einstein notation, this can be written more compactly as

$$C_{\{i_1, i_2, \dots, i_a\} \setminus \{i_x, i_y\}} = A_{i_1, i_2, \dots, i_a} \delta^{i_x, i_y}$$

where δ is the Kronecker delta. In general, the contraction of A along dimensions $\{x_1, \dots, x_p\}$ yields a tensor $C = A_{\downarrow x_1, \dots, x_p} \in \mathbb{R}^{n^{a-p}}$ where

$$A_{\downarrow x_1, \dots, x_p} = \sum_{i_{x_1}} \sum_{i_{x_2}} \dots \sum_{i_{x_p}} A_{i_1, i_2, \dots, i_a}$$

or compactly as

$$A_{\downarrow x_1, \dots, x_p} = A_{i_1, i_2, \dots, i_a} \delta^{i_{x_1}, i_{x_2}, \dots, i_{x_p}}.$$

Based on these tensor contractions and Def. 4.3.5, we can construct the second-order \mathbb{S}_n -equivariant networks as in Def. 4.3.6 (see Example 4.3.1): $f = \gamma \circ \mathcal{M}_T \circ \dots \circ \gamma \circ \mathcal{M}_1$. The second-order networks are particularly essential for us to extend the original variational autoencoder (VAE) (Kingma and Welling, 2014) model that approximates the posterior distribution by an *isotropic* Gaussian distribution with a diagonal covariance matrix and uses a fixed prior distribution $\mathcal{N}(0, 1)$. In contrast, we generalize by modeling the posterior by $\mathcal{N}(\mu, \Sigma)$ in which Σ is a full covariance matrix, and we learn an adaptive parameterized prior $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$ instead of a fixed one. Only the second-order encoders can output a permutation equivariant full covariance matrix, while lower-order networks such as MPNNs are unable to. See Sec. 4.4.2, 4.5 and 4.6 for details.

Example 4.3.1. *The second order message passing has the message $\mathcal{H}_0 \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}| \times (d_v + d_e)}$ initialized by promoting the node features \mathcal{F}_v to a second order tensor (e.g., we treat node features as self-loop edge features), and concatenating with the edge features \mathcal{F}_e . Iteratively,*

$$\mathcal{H}_t = \gamma(\mathcal{M}_t), \quad \mathcal{M}_t = \mathcal{W}_t \left[\bigoplus_{i,j} (\mathcal{A} \otimes \mathcal{H}_{t-1})_{\downarrow i,j} \right],$$

where $\mathcal{A} \otimes \mathcal{H}_{t-1}$ results in a fourth order tensor while $\downarrow_{i,j}$ contracts it down to a second order tensor along the i -th and j -th dimensions, \oplus denotes concatenation along the feature channels, and \mathcal{W}_t denotes a multilayer perceptron on the feature channels. We remark that the popular MPNNs (Gilmer et al., 2017) is a lower-order one and a special case in which $\mathcal{M}_t = \mathcal{D}^{-1} \mathcal{A} \mathcal{H}_{t-1} \mathcal{W}_{t-1}$ where $\mathcal{D}_{ii} = \sum_j \mathcal{A}_{ij}$ is the diagonal matrix of node degrees. The message \mathcal{H}_T of the last iteration is still second order, so we contract it down to the first order latent $\mathcal{Z} = \bigoplus_i \mathcal{H}_{T \downarrow i}$.

4.3.3 Learning to cluster

Definition 4.3.9. A clustering of n objects into k clusters is a mapping $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, k\}$ in which $\pi(i) = j$ if the i -th object is assigned to the j -th cluster. The inverse mapping $\pi^{-1}(j) = \{i \in [1, n] : \pi(i) = j\}$ gives the set of all objects assigned to the j -th cluster. The clustering is represented by an assignment matrix $\Pi \in \{0, 1\}^{n \times k}$ such that $\Pi_{i, \pi(i)} = 1$.

Definition 4.3.10. The action of \mathbb{S}_n on a clustering π of n objects into k clusters and its corresponding assignment matrix Π are

$$[\sigma \cdot \pi](i) = \pi(\sigma^{-1}(i)), \quad [\sigma \cdot \Pi]_{i,j} = \Pi_{\sigma^{-1}(i), j}, \quad \sigma \in \mathbb{S}_n.$$

Definition 4.3.11. Let \mathcal{N} be a neural network that takes as input a graph \mathcal{G} of n nodes, and outputs a clustering π of k clusters. \mathcal{N} is said to be *equivariant* if and only if $\mathcal{N}(\sigma \cdot \mathcal{G}) = \sigma \cdot \mathcal{N}(\mathcal{G})$ for all $\sigma \in \mathbb{S}_n$.

From Def. 4.3.11, intuitively the assignment matrix Π still represents the same clustering if we permute its rows. The learnable clustering procedure $\mathbf{c}(\mathcal{G}^{(\ell)}; \boldsymbol{\theta})$ is built as follows:

1. A graph neural network parameterized by $\boldsymbol{\theta}$ encodes graph $\mathcal{G}^{(\ell)}$ into a first order tensor of K feature channels $\tilde{p}^{(\ell)} \in \mathbb{R}^{|\mathcal{V}^{(\ell)}| \times K}$.
2. The clustering assignment is determined by a row-wise maximum pooling operation:

$$\pi^{(\ell)}(i) = \arg \max_{k \in [1, K]} \tilde{p}_{i,k}^{(\ell)} \tag{4.1}$$

that is an equivariant clustering in the sense of Def. 4.3.11.

A composition of an equivariant function (e.g., graph net) and an equivariant function (e.g., maximum pooling given in Eq. 4.1) is still an equivariant function with respect to the node permutation. Thus, the learnable clustering procedure $\mathbf{c}(\mathcal{G}^{(\ell)}; \boldsymbol{\theta})$ is permutation equivariant.

In practice, in order to make the clustering procedure differentiable for backpropagation, we replace the maximum pooling in Eq. 4.1 by sampling from a categorical distribution. Let $\pi^{(\ell)}(i)$ be a categorical variable with class probabilities $p_{i,1}^{(\ell)}, \dots, p_{i,K}^{(\ell)}$ computed as softmax from $\tilde{p}_{i,:}^{(\ell)}$. The Gumbel-max trick (Gumbel, 1954)(Maddison et al., 2014)(Jang et al., 2017) provides a simple and efficient way to draw samples $\pi^{(\ell)}(i)$:

$$\Pi_i^{(\ell)} = \text{one-hot}\left(\arg\max_{k \in [1, K]} [g_{i,k} + \log p_{i,k}^{(\ell)}]\right),$$

where $g_{i,1}, \dots, g_{i,K}$ are i.i.d samples drawn from $\text{Gumbel}(0, 1)$. Given the clustering assignment matrix $\Pi^{(\ell)}$, the coarsened adjacency matrix $\mathcal{A}^{(\ell-1)}$ (see Defs. 4.3.1 and 4.3.2) can be constructed as $\Pi^{(\ell)}^T \mathcal{A}^{(\ell)} \Pi^{(\ell)}$.

It is desirable to have a *balanced* K -cluster partition in which clusters $\mathcal{V}_1^{(\ell)}, \dots, \mathcal{V}_K^{(\ell)}$ have similar sizes that are close to $|\mathcal{V}^{(\ell)}|/K$. The local encoders tend to generalize better for same-size subgraphs. We want the distribution of nodes into clusters to be close to the uniform distribution. We enforce the clustering procedure to produce a balanced cut by minimizing the following Kullback–Leibler divergence:

$$\mathcal{D}_{KL}(P||Q) = \sum_{k=1}^K P(k) \log \frac{P(k)}{Q(k)} \quad \text{where} \quad P = \left(\frac{|\mathcal{V}_1^{(\ell)}|}{|\mathcal{V}^{(\ell)}|}, \dots, \frac{|\mathcal{V}_K^{(\ell)}|}{|\mathcal{V}^{(\ell)}|} \right), \quad Q = \left(\frac{1}{K}, \dots, \frac{1}{K} \right). \quad (4.2)$$

The whole construction of MGN is *equivariant* with respect to node permutations of \mathcal{G} . In the case of molecular property prediction, we want MGN to learn to predict a real value $y \in \mathbb{R}$ for each graph \mathcal{G} while learning to find a balanced cut in each resolution to construct a hierarchical structure of latents and coarsen graphs. The total loss function is

$$\mathcal{L}_{\text{MGN}}(\mathcal{G}, y) = \left\| f\left(\bigoplus_{\ell=1}^L R(\mathcal{Z}^{(\ell)})\right) - y \right\|_2^2 + \sum_{\ell=1}^L \lambda^{(\ell)} \mathcal{D}_{\text{KL}}(P^{(\ell)}||Q^{(\ell)}), \quad (4.3)$$

where f is a multilayer perceptron, \oplus denotes the vector concatenation, R is a readout function that produces a permutation invariant vector of size d given the latent $\mathcal{Z}^{|\mathcal{V}^{(\ell)}| \times d}$ at the ℓ -th resolution, $\lambda^{(\ell)} \in \mathbb{R}$ is a hyperparameter, and $\mathcal{D}_{\text{KL}}(P^{(\ell)} \| Q^{(\ell)})$ is the balanced-cut loss as defined in Eq. 4.2.

4.4 Hierarchical generative model

In this section, we introduce our hierarchical generative model for multiresolution graph generation based on variational principles.

4.4.1 Background on graph variational autoencoder

Suppose that we have input data consisting of m graphs (data points) $\mathcal{G} = \{\mathcal{G}_1, \dots, \mathcal{G}_m\}$. The standard variational autoencoders (VAEs), introduced by Kingma and Welling (2014) have the following generation process, in which each data graph \mathcal{G}_i for $i \in \{1, 2, \dots, m\}$ is generated independently:

1. Generate the latent variables $\mathcal{Z} = \{\mathcal{Z}_1, \dots, \mathcal{Z}_m\}$, where each $\mathcal{Z}_i \in \mathbb{R}^{|\mathcal{V}_i| \times d_z}$ is drawn i.i.d. from a prior distribution p_0 (e.g., standard Normal distribution $\mathcal{N}(0, 1)$).
2. Generate the data graph $\mathcal{G}_i \sim p_\theta(\mathcal{G}_i | \mathcal{Z}_i)$ from the model conditional distribution p_θ .

We want to optimize θ to maximize the likelihood $p_\theta(\mathcal{G}) = \int p_\theta(\mathcal{Z}) p_\theta(\mathcal{G} | \mathcal{Z}) d\mathcal{Z}$. However, this requires computing the posterior distribution $p_\theta(\mathcal{G} | \mathcal{Z}) = \prod_{i=1}^m p_\theta(\mathcal{G}_i | \mathcal{Z}_i)$, which is usually intractable. Instead, VAEs apply the variational principle, proposed by Wainwright and Jordan (2005), to approximate the posterior distribution as $q_\phi(\mathcal{Z} | \mathcal{G}) = \prod_{i=1}^m q_\phi(\mathcal{Z}_i | \mathcal{G}_i)$ via amortized inference and maximize the *evidence lower bound* (ELBO) that is a lower bound of the likelihood:

$$\begin{aligned} \mathcal{L}_{\text{ELBO}}(\phi, \theta) &= \mathbb{E}_{q_\phi(\mathcal{Z} | \mathcal{G})} [\log p_\theta(\mathcal{G} | \mathcal{Z})] - \mathcal{D}_{\text{KL}}(q_\phi(\mathcal{Z} | \mathcal{G}) \| p_0(\mathcal{Z})) \\ &= \sum_{i=1}^m \left[\mathbb{E}_{q_\phi(\mathcal{Z}_i | \mathcal{G}_i)} [\log p_\theta(\mathcal{G}_i | \mathcal{Z}_i)] - \mathcal{D}_{\text{KL}}(q_\phi(\mathcal{Z}_i | \mathcal{G}_i) \| p_0(\mathcal{Z}_i)) \right]. \end{aligned} \quad (4.4)$$

The probabilistic encoder $q_\phi(\mathcal{Z}|\mathcal{G})$, the approximation to the posterior of the generative model $p_\theta(\mathcal{G}, \mathcal{Z})$, is modeled using equivariant graph neural networks (see Example 4.3.1) as follows. Assume the prior over the latent variables to be the centered isotropic multivariate Gaussian $p_\theta(\mathcal{Z}) = \mathcal{N}(\mathcal{Z}; 0, I)$. We let $q_\phi(\mathcal{Z}_i|\mathcal{G}_i)$ be a multivariate Gaussian with a diagonal covariance structure:

$$\log q_\phi(\mathcal{Z}_i|\mathcal{G}_i) = \log \mathcal{N}(\mathcal{Z}_i; \mu_i, \sigma_i^2 I), \quad (4.5)$$

where $\mu_i, \sigma_i \in \mathbb{R}^{|\mathcal{V}_i| \times d_z}$ are the mean and standard deviation of the approximate posterior output by two equivariant graph encoders. We sample from the posterior q_ϕ by using the reparameterization trick: $\mathcal{Z}_i = \mu_i + \sigma_i \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$ and \odot is the element-wise product.

On the other hand, the probabilistic decoder $p_\theta(\mathcal{G}_i|\mathcal{Z}_i)$ defines a conditional distribution over the entries of the adjacency matrix \mathcal{A}_i : $p_\theta(\mathcal{G}_i|\mathcal{Z}_i) = \prod_{(u,v) \in \mathcal{V}_i^2} p_\theta(\mathcal{A}_{iuv} = 1 | \mathcal{Z}_{iu}, \mathcal{Z}_{iv})$. For example, Kipf and Welling (2016) suggests a simple dot-product decoder that is trivially equivariant: $p_\theta(\mathcal{A}_{iuv} = 1 | \mathcal{Z}_{iu}, \mathcal{Z}_{iv}) = \gamma(\mathcal{Z}_{iu}^T \mathcal{Z}_{iv})$, where γ denotes the sigmoid function.

4.4.2 Multiresolution VAEs

Based on the construction of multiresolution graph network (see Sec. 4.3.1), the latent variables are partitioned into disjoint groups, $\mathcal{Z}_i = \{\mathcal{Z}_i^{(1)}, \mathcal{Z}_i^{(2)}, \dots, \mathcal{Z}_i^{(L)}\}$ where $\mathcal{Z}_i^{(\ell)} = \{[\mathcal{Z}_i^{(\ell)}]_k \in \mathbb{R}^{|\mathcal{V}_i^{(\ell)}|_k \times d_z}\}_k$ is the set of latents at the ℓ -th resolution level in which the graph $\mathcal{G}_i^{(\ell)}$ is partitioned into a number of clusters $[\mathcal{G}_i^{(\ell)}]_k$.

In the area of normalizing flows (NFs), Wu et al. (2020) has shown that stochasticity (e.g., a chain of stochastic sampling blocks) overcomes expressivity limitations of NFs. In general, our MGVAE is a stochastic version of the deterministic MGN such that stochastic sampling is applied at each resolution level in a bottom-up manner. The prior (Eq. 4.6) and the

approximate posterior (Eq. 4.7) are represented by

$$p(\mathcal{Z}_i) = \prod_{\ell=1}^L p(\mathcal{Z}_i^{(\ell)}) = \prod_{\ell=1}^L \prod_k p([\mathcal{Z}_i^{(\ell)}]_k), \quad (4.6)$$

$$q_\phi(\mathcal{Z}_i | \mathcal{G}_i) = q_\phi(\mathcal{Z}_i^{(L)} | \mathcal{G}_i^{(L)}) \prod_{\ell=L-1}^1 q_\phi(\mathcal{Z}_i^{(\ell)} | \mathcal{Z}_i^{(\ell+1)}, \mathcal{G}_i^{(\ell)}), \quad (4.7)$$

in which each conditional in the approximate posterior are in the form of factorial Normal distributions, in particular

$$q_\phi(\mathcal{Z}_i^{(\ell)} | \mathcal{Z}_i^{(\ell+1)}, \mathcal{G}_i^{(\ell)}) = \prod_k q_\phi([\mathcal{Z}_i^{(\ell)}]_k | \mathcal{Z}_i^{(\ell+1)}, [\mathcal{G}_i^{(\ell)}]_k),$$

where each probabilistic encoder $q_\phi([\mathcal{Z}_i^{(\ell)}]_k | \mathcal{Z}_i^{(\ell+1)}, [\mathcal{G}_i^{(\ell)}]_k)$ operates on a subgraph $[\mathcal{G}_i^{(\ell)}]_k$ as follows:

- The pooling network $\mathbf{p}^{(\ell+1)}$ shrinks the latent $\mathcal{Z}_i^{(\ell+1)}$ into the node features of $\mathcal{G}_i^{(\ell)}$ as in the construction of MGN (see Def. 4.3.4).
- The local (deterministic) graph encoder $\mathbf{d}_{\text{local}}^{(\ell)}$ encodes each subgraph $[\mathcal{G}_i^{(\ell)}]_k$ into a mean vector and a diagonal covariance matrix (see Eq. 4.5). A second order encoder can produce a positive semidefinite non-diagonal covariance matrix, that can be interpreted as a Gaussian Markov Random Fields (details in Sec. 4.5). The new subgraph latent $[\mathcal{Z}_i^{(\ell)}]_k$ is sampled by the reparameterization trick.

The prior can be either the isotropic Gaussian $\mathcal{N}(0, 1)$ as in standard VAEs, or be implemented as a parameterized Gaussian $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$ where $\hat{\mu}$ and $\hat{\Sigma}$ are learnable equivariant functions (details in Sec. 4.6). The reparameterization trick for conventional $\mathcal{N}(0, 1)$ prior is the same as in Sec. 4.4.1, while the new one for the generalized and learnable prior $\mathcal{N}(\hat{\mu}, \hat{\Sigma})$ is given in Sec. 4.5. On the another hand, the probabilistic decoder $p_\theta(\mathcal{G}_i^{(1)}, \dots, \mathcal{G}_i^{(L)} | \mathcal{Z}_i^{(1)}, \dots, \mathcal{Z}_i^{(L)})$

defines a conditional distribution over all subgraph adjacencies at each resolution level:

$$p_\theta(\mathcal{G}_i^{(1)}, \dots, \mathcal{G}_i^{(L)} | \mathcal{Z}_i^{(1)}, \dots, \mathcal{Z}_i^{(L)}) = \prod_{\ell} p_\theta(\mathcal{G}_i^{(\ell)} | \mathcal{Z}_i^{(\ell)}) = \prod_{\ell} \prod_k p_\theta([\mathcal{A}_i^{(\ell)}]_k | [\mathcal{Z}_i^{(\ell)}]_k).$$

Extending from Eq. 4.4, we write our multiresolution variational lower bound $\mathcal{L}_{\text{MGVAE}}(\phi, \theta)$ on $\log p(\mathcal{G})$ compactly as

$$\mathcal{L}_{\text{MGVAE}}(\phi, \theta) = \sum_i \sum_{\ell} \left[\mathbb{E}_{q_\phi(\mathcal{Z}_i^{(\ell)} | \mathcal{G}_i^{(\ell)})} [\log p_\theta(\mathcal{G}_i^{(\ell)} | \mathcal{Z}_i^{(\ell)})] - \mathcal{D}_{\text{KL}}(q_\phi(\mathcal{Z}_i^{(\ell)} | \mathcal{G}_i^{(\ell)}) \| p_0(\mathcal{Z}_i^{(\ell)})) \right], \quad (4.8)$$

where the first term denotes the reconstruction loss (e.g., $\|\mathcal{A}_i^{(\ell)} - \hat{\mathcal{A}}_i^{(\ell)}\|$ where $\mathcal{A}_i^{(L)}$ is \mathcal{G}_i itself, $\mathcal{A}_i^{(\ell < L)}$ is the adjacency produced by MGN at level ℓ , and $\hat{\mathcal{A}}_i^{(\ell)}$ are the reconstructed ones by the decoders); and the second term is indeed $\mathcal{D}_{\text{KL}}(\mathcal{N}(\mu_i^{(\ell)}, \Sigma_i^{(\ell)}) \| \mathcal{N}(\hat{\mu}^{(\ell)}, \hat{\Sigma}^{(\ell)}))$ where $\mu_i^{(\ell)} \in \mathbb{R}^{|\mathcal{V}_i^{(\ell)}| \times d}$ and $\Sigma_i^{(\ell)} \in \mathbb{R}^{|\mathcal{V}_i^{(\ell)}| \times |\mathcal{V}_i^{(\ell)}| \times d}$ are the mean and covariance tensors produced by the ℓ -th encoder for graph \mathcal{G}_i , while $\hat{\mu}^{(\ell)}$ and $\hat{\Sigma}^{(\ell)}$ are learnable ones in an equivariant manner as in Sec. 4.6. In general, the overall optimization is given as follows:

$$\min_{\phi, \theta, \{\hat{\mu}^{(\ell)}, \hat{\Sigma}^{(\ell)}\}_{\ell}} \mathcal{L}_{\text{MGVAE}}(\phi, \theta; \{\hat{\mu}^{(\ell)}, \hat{\Sigma}^{(\ell)}\}_{\ell}) + \sum_{i, \ell} \lambda^{(\ell)} \mathcal{D}_{\text{KL}}(P_i^{(\ell)} \| Q_i^{(\ell)}), \quad (4.9)$$

where ϕ denotes all learnable parameters of the encoders, θ denotes all learnable parameters of the decoders, and $\mathcal{D}_{\text{KL}}(P_i^{(\ell)} \| Q_i^{(\ell)})$ is the balanced-cut loss for graph \mathcal{G}_i at level ℓ as defined in Sec. 4.3.3.

4.5 Markov Random Fields

Undirected graphical models have been widely applied in the domains spatial or relational data, such as image analysis and spatial statistics. In general, k -th order graph encoders encode an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ into a k -th order latent $\mathbf{z} \in \mathbb{R}^{n^k \times d_z}$, with learnable parameters $\boldsymbol{\theta}$, can be represented as a parameterized Markov Random Field (MRF) or Markov

network. Based on the Hammersley-Clifford theorem (Murphy, 2012a) (Koller and Friedman, 2009), a positive distribution $p(\mathbf{z}) > 0$ satisfies the conditional independent properties of an undirected graph \mathcal{G} iff p can be represented as a product of potential functions ψ , one per *maximal clique*, i.e.,

$$p(\mathbf{z}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{c \in \mathcal{C}} \psi_c(z_c|\theta_c) \quad (4.10)$$

where \mathcal{C} is the set of all the (maximal) cliques of \mathcal{G} , and $Z(\boldsymbol{\theta})$ is the *partition function* to ensure the overall distribution sums to 1, and given by

$$Z(\boldsymbol{\theta}) = \sum_{\mathbf{z}} \prod_{c \in \mathcal{C}} \psi_c(z_c|\theta_c)$$

Eq. 4.10 can be further written down as

$$p(\mathbf{z}|\boldsymbol{\theta}) \propto \prod_{v \in \mathcal{V}} \psi_v(z_v|\boldsymbol{\theta}) \prod_{(s,t) \in \mathcal{E}} \psi_{st}(z_{st}|\boldsymbol{\theta}) \cdots \prod_{c=(i_1, \dots, i_k) \in \mathcal{C}_k} \psi_c(z_c|\boldsymbol{\theta})$$

where ψ_v , ψ_{st} , and ψ_c are the first order, second order and k -th order outputs of the encoder, corresponding to every vertex in \mathcal{V} , every edge in \mathcal{E} and every clique of size k in \mathcal{C}_k , respectively. However, factorizing a graph into set of maximal cliques has an exponential time complexity, since the problem of determining if there is a clique of size k in a graph is known as an NP-complete problem. Thus, the factorization based on Hammersley-Clifford theorem is *intractable*. The second order encoder relaxes the restriction of maximal clique into *edges*, that is called as *pairwise MRF*:

$$p(\mathbf{z}|\boldsymbol{\theta}) \propto \prod_{s \sim t} \psi_{st}(z_s, z_t)$$

Our second order encoder inherits Gaussian MRF introduced by (Rue and Held, 2005) as *pairwise* MRF of the following form

$$p(\mathbf{z}|\boldsymbol{\theta}) \propto \prod_{s \sim t} \psi_{st}(z_s, z_t) \prod_t \psi_t(z_t)$$

where $\psi_{st}(z_s, z_t) = \exp\left(-\frac{1}{2}z_s \Lambda_{st} z_t\right)$ is the edge potential, and $\psi_t(z_t) = \exp\left(-\frac{1}{2}\Lambda_{tt} z_t^2 + \eta_t z_t\right)$ is the vertex potential. The joint distribution can be written in the *information form* of a multivariate Gaussian in which

$$\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$$

$$\boldsymbol{\eta} = \boldsymbol{\Lambda} \boldsymbol{\mu}$$

$$p(\mathbf{z}|\boldsymbol{\theta}) \propto \exp\left(\boldsymbol{\eta}^T \mathbf{z} - \frac{1}{2} \mathbf{z}^T \boldsymbol{\Lambda} \mathbf{z}\right) \quad (4.11)$$

Sampling \mathbf{z} from $p(\mathbf{z}|\boldsymbol{\theta})$ in Eq. 4.11 is the same as sampling from the multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. To ensure end-to-end equivariance, we set the latent layer to be two tensors $\boldsymbol{\mu} \in \mathbb{R}^{n \times d_z}$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n \times d_z}$ that corresponds to d_z multivariate Gaussians, whose first index, and second index are first order and second order equivariant with permutations. Computation of $\boldsymbol{\Sigma}$ is trickier than $\boldsymbol{\mu}$, simply because $\boldsymbol{\Sigma}$ must be invertible to be a covariance matrix. Thus, our second order encoder produces tensor \mathbf{L} as the second order activation, and set $\boldsymbol{\Sigma} = \mathbf{L} \mathbf{L}^T$. The reparameterization trick from Kingma and Welling (2014) is changed to

$$\mathbf{z} = \boldsymbol{\mu} + \mathbf{L} \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, 1)$$

4.6 Equivariant learnable prior

The original VAE published by Kingma and Welling (2014) limits each covariance matrix $\boldsymbol{\Sigma}$ to be diagonal and the prior to be $\mathcal{N}(0, 1)$. Our second order encoder removes the diagonal restriction on the covariance matrix. Furthermore, we allow the prior $\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$

to be *learnable* in which $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ are parameters optimized by back propagation in a data driven manner. Importantly, $\hat{\boldsymbol{\Sigma}}$ cannot be learned directly due to the invertibility restriction. Instead, similarly to the second order encoder, a matrix $\hat{\mathbf{L}}$ is optimized, and the prior covariance matrix is constructed by setting $\hat{\boldsymbol{\Sigma}} = \hat{\mathbf{L}}\hat{\mathbf{L}}^T$. The Kullback-Leibler divergence between the two distributions $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$ is as follows:

$$\mathcal{D}_{KL}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \parallel \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})) = \frac{1}{2} \left(\text{tr}(\hat{\boldsymbol{\Sigma}}^{-1} \boldsymbol{\Sigma}) + (\hat{\boldsymbol{\mu}} - \boldsymbol{\mu})^T \hat{\boldsymbol{\Sigma}}^{-1} (\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}) - n + \ln \left(\frac{\det \hat{\boldsymbol{\Sigma}}}{\det \boldsymbol{\Sigma}} \right) \right) \quad (4.12)$$

Even though $\boldsymbol{\Sigma}$ is invertible, but gradient computation through the KL-divergence loss can be numerical unstable because of Cholesky decomposition procedure in matrix inversion. Thus, we add neglectable noise $\epsilon = 10^{-4}$ to the diagonal of both covariance matrices.

Importantly, during training, the KL-divergence loss breaks the permutation equivariance. Suppose the set of vertices are permuted by a permutation matrix \mathbf{P}_σ for $\sigma \in \mathbb{S}_n$. Since $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the first order and second order equivariant outputs of the encoder, they are changed to $\mathbf{P}_\sigma \boldsymbol{\mu}$ and $\mathbf{P}_\sigma \boldsymbol{\Sigma} \mathbf{P}_\sigma^T$ accordingly. But

$$\mathcal{D}_{KL}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \parallel \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})) \neq \mathcal{D}_{KL}(\mathcal{N}(\mathbf{P}_\sigma \boldsymbol{\mu}, \mathbf{P}_\sigma \boldsymbol{\Sigma} \mathbf{P}_\sigma^T) \parallel \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}))$$

To address the equivariance issue, we want to solve the following convex optimization problem that is our new equivariant loss function

$$\min_{\sigma \in \mathbb{S}_n} \mathcal{D}_{KL}(\mathcal{N}(\mathbf{P}_\sigma \boldsymbol{\mu}, \mathbf{P}_\sigma \boldsymbol{\Sigma} \mathbf{P}_\sigma^T) \parallel \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})) \quad (4.13)$$

However, solving the optimization based on Eq. 4.13 is computationally expensive. One solution is to solve the minimum-cost maximum-matching in a bipartite graph (Hungarian matching) with the cost matrix $\mathbf{C}_{ij} = \|\boldsymbol{\mu}_i - \hat{\boldsymbol{\mu}}_j\|$ by $O(n^4)$ algorithm published by Edmonds

and Karp (1972), that can be still improved further into $O(n^3)$. The Hungarian matching preserves equiariance, but is still computationally expensive. In practice, instead of finding a optimal permutation, we apply a free-matching scheme to find an assignment matrix Π such that: $\Pi_{ij^*} = 1$ if and only if $j^* = \arg \min_j \|\mu_i - \hat{\mu}_j\|$, for each $i \in [1, n]$. The free-matching scheme preserves equivariance and can be done efficiently in a simple $O(n^2)$ algorithm that is also suitable for GPU computation.

4.7 Experiments

4.7.1 Molecular graph generation

We examine the generative power of MGN and MGVAE in the challenging task of molecule generation, in which the graphs are highly structured. We demonstrate that MGVAE is the first hierarchical graph VAE model generating graphs in a permutation-equivariant manner that is competitive against autoregressive results. We train on two datasets that are standard in the field:

1. **QM9** (Ruddigkeit et al., 2012) (Ramakrishnan et al., 2014): contains 134K organic molecules with up to nine atoms (C, H, O, N, and F) out of the GDB-17 universe of molecules.
2. **ZINC** (Sterling and Irwin, 2015): contains 250K purchasable drug-like chemical compounds with up to twenty-three heavy atoms.

We only use the graph features as the input, including the adjacency matrix, the one-hot vector of atom types (e.g., carbon, hydrogen, etc.) and the bond types (single bond, double bond, etc.) without any further domain knowledge from chemistry or physics. First, we train autoencoding task of reconstructing the adjacency matrix and node features. We use a learnable equivariant prior (see Sec. 4.6) instead of the conventional $\mathcal{N}(0, 1)$. Then, we generate 5,000 different samples from the prior, and decode each sample into a generated

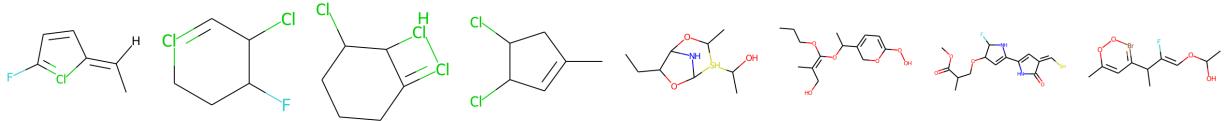


Figure 4.3: MGVAE generates molecules on QM9 (4 on the left) and ZINC (the rest) equivariantly. Both equivariant MGVAE and autoregressive MGN generate high-quality molecules with complicated structures such as rings.

graph (see Fig. 4.3). We implement our graph construction (decoding) in two approaches:

1. **All-at-once:** We reconstruct the whole adjacency matrix by running the probabilistic decoder (see Sec. 4.4). MGVAE enables us to generate a graph at any given resolution level ℓ . In this particular case, we select the highest resolution $\ell = L$. Furthermore, we apply learnable equivariant prior as in Sec. 4.6. Our second order encoders are interpreted as Markov Random Fields (see Sec. 4.5). This approach preserves permutation equivariance. In addition, we implement a correcting process: the decoder network of the highest resolution level returns a probability for each edge, we sort these probabilities in a descending order and gradually add the edges in that order to satisfy all chemical constraints. Furthermore, we investigate the expressive power of the second order \mathbb{S}_n -equivariant decoder by replacing it by a multilayer perceptron (MLP) decoder with 2 hidden layers of size 512 and sigmoid nonlinearity. We find that the higher order decoder outperforms the MLP decoder given the same encoding architecture. Table 4.2 shows the comparison between the two decoding models.
2. **Autoregressive:** This decoding process is constructed in an autoregressive manner similarly to (Liu et al., 2018). First, we sample each vertex latent z independently. We randomly select a starting node v_0 , then we apply Breath First Search (BFS) to determine a particular node ordering from the node v_0 , however that breaks the permutation equivariance. Then iteratively we add/sample new edge to the existing graph \mathcal{G}_t at the t -th iteration (given a randomly selected node v_0 as the start graph

Dataset	Method	Train size	Features	Validity	Novelty	Uniqueness	
QM9	GraphVAE	~ 100K	Graph	61.00%	85.00%	40.90%	
	CGVAE			100%	94.35%	98.57%	
	MolGAN			98.1%	94.2%	10.4%	
	Autoregressive MGN	10K		100%	95.01%	97.44%	
	All-at-once MGVAE			100%	100%	95.16%	
ZINC	GraphVAE	~ 200K	Graph	14.00%	100%	31.60%	
	CGVAE			100%	100%	99.82%	
	JT-VAE			100%	-	-	
	Autoregressive MGN	1K		100%	99.89%	99.69%	
	All-at-once MGVAE	10K	Chemical	99.92%	100%	99.34%	

Table 4.1: Molecular graph generation results. GraphVAE results are taken from (Liu et al., 2018).

\mathcal{G}_0) until completion. We apply second-order MGN with gated recurrent architecture to produce the probability of edge (u, v) where one vertex u is in the existing graph \mathcal{G}_t and the another one is outside; and also the probability of its label. Intuitively, the decoding process is a sequential classification.

In our setting for small molecules, $L = 3$ and $K = 2^{\ell-1}$ for the ℓ -th level. On each resolution level, the local encoders and local decoders are second-order \mathbb{S}_n -equivariant networks with up to 4 equivariant layers. The number of channels for each node latent d_z is set to 256. We compare our methods with other graph-based generative models including GraphVAE (Simonovsky and Komodakis, 2018), CGVAE (Liu et al., 2018), MolGAN (Cao and Kipf, 2018), and JT-VAE (Jin et al., 2018). We evaluate the quality of generated molecules in three metrics: (i) validity, (ii) novelty and (iii) uniqueness as the percentage of the generated molecules that are chemically valid, different from all molecules in the training set, and not redundant, respectively.

We randomly select 10,000 training examples for QM9; and 1,000 (autoregressive) and 10,000 (all-at-once) training examples for ZINC. It is important to note that our training sets are much smaller comparing to other methods. For all of our generation experiments, we only

use graph features as the input for the encoder such as one-hot atomic types and bond types. Since ZINC molecules are larger than QM9 ones, it is more difficult to train with the second order S_n -equivariant decoders (e.g., the number of bond/non-bond predictions or the number of entries in the adjacency matrices are proportional to squared number of nodes). Therefore, we input several chemical/atomic features computed from RDKit for the all-at-once MGVAE on ZINC (see Table 4.3). We concatenate all these features into a vector of size 24 for each atom.

We train our models with Adam optimization method (Kingma and Ba, 2015) with the initial learning rate of 10^{-3} . Figs. 4.4 and 4.5 show some selected examples out of 5,000 generated molecules on QM9 by all-at-once MGVAE, while Fig. 4.6 shows the molecules generated by autoregressive MGN. Qualitatively, both the decoding approaches capture similar molecular substructures (bond structures). Fig. 4.9 shows an example of interpolation on the latent space on ZINC with the all-at-once MGVAE. Fig. 4.7 shows some generated molecules on ZINC by the all-at-once MGVAE. Fig. 4.8 and table 4.4 show some generated molecules by the autoregressive MGN on ZINC dataset with high Quantitative Estimate of Drug-Likeness (QED) computed by RDKit and their SMILES strings. On ZINC, the average QED score of the generated molecules is 0.45 with standard deviation 0.21. On QM9, the QED score is 0.44 ± 0.07 .

Our models are equivalent with the state-of-the-art, even with a limited training set (see Table 4.1). Figure 4.3 shows some randomly selected examples out of 5,000 generated molecules. Admittedly, molecule generation is a somewhat subjective task that can only be evaluated with objective numerical measures up to a certain point. Qualitatively, however the molecules that MGVAE generates are as good as the state of the art, in some cases better in terms of producing several high-quality drug-like molecules with complicated functional

Dataset	Method	Validity	Novelty	Uniqueness
QM9	MLP decoder	100%	99.98%	77.62%
	\mathbb{S}_n decoder	100%	100%	95.16%

Table 4.2: All-at-once MGVAE with MLP decoder vs. second order decoder.

Feature	Type	Number	Description
GetAtomicNum	Integer	1	Atomic number
IsInRing	Boolean	1	Belongs to a ring?
IsInRingSize	Boolean	9	Belongs to a ring of size $k \in \{1, \dots, 9\}$?
GetIsAromatic	Boolean	1	Aromaticity?
GetDegree	Integer	1	Vertex degree
GetExplicitValance	Integer	1	Explicit valance
GetFormalCharge	Integer	1	Formal charge
GetIsotope	Integer	1	Isotope
GetMass	Double	1	Atomic mass
GetNoImplicit	Boolean	1	Allowed to have implicit Hs?
GetNumExplicitHs	Integer	1	Number of explicit Hs
GetNumImplicitHs	Integer	1	Number of implicit Hs
GetNumRadicalElectrons	Integer	1	Number of radical electrons
GetTotalDegree	Integer	1	Total degree
GetTotalNumHs	Integer	1	Total number of Hs
GetTotalValence	Integer	1	Total valance

Table 4.3: The list of chemical/atomic features used for the all-at-once MGVAE on ZINC. We denote each feature by its API in RDKit.

groups and structures.

4.7.2 Unsupervised molecular properties prediction on QM9

Density Function Theory (DFT) is the most successful and widely used approach of modern quantum chemistry to compute the electronic structure of matter, and to calculate many properties of molecular systems with high accuracy (Hohenberg and Kohn, 1964). However, DFT is computationally expensive (Gilmer et al., 2017), that leads to the use of machine learning to estimate the properties of compounds from their chemical structure rather than computing them explicitly with DFT (Hy et al., 2018). To demonstrate that MGVAE can learn a useful molecular representations and capture important molecular structures in an

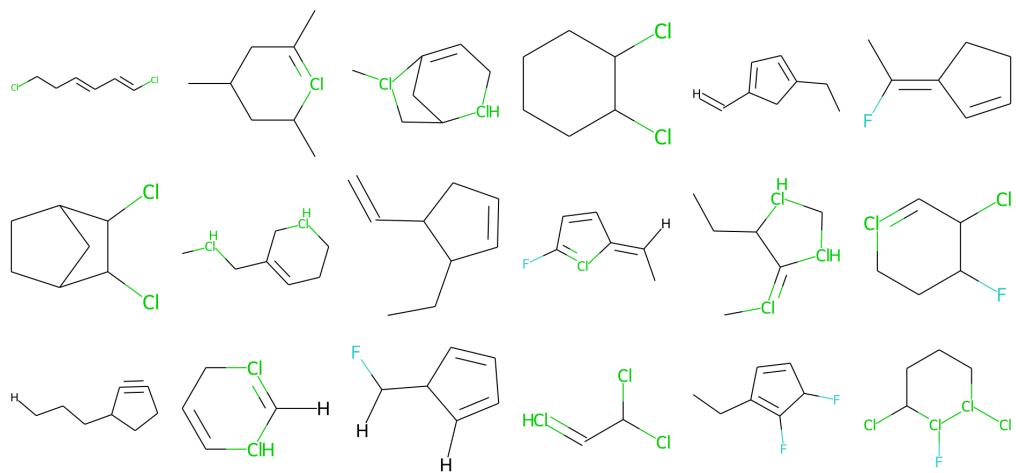


Figure 4.4: Some generated examples on QM9 by the all-at-once MGVAE with second order \mathbb{S}_n -equivariant decoders.

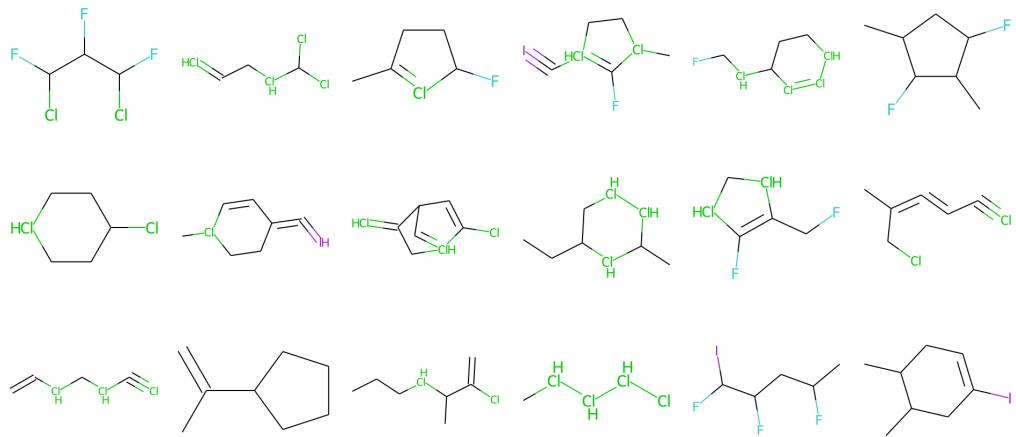


Figure 4.5: Some generated examples on QM9 by the all-at-once MGVAE with a MLP decoder instead of the second order \mathbb{S}_n -equivariant one. It generates more tree-like structures.

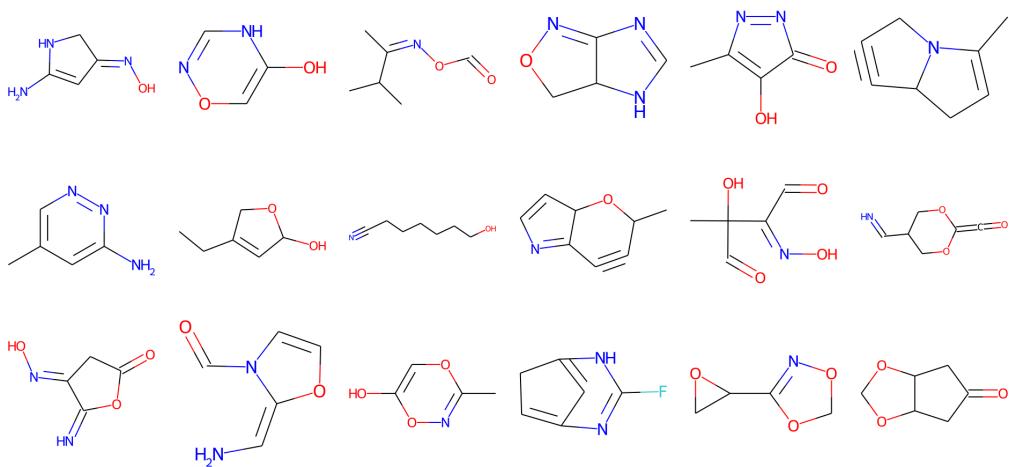


Figure 4.6: Some generated examples on QM9 by the autoregressive MGN.

Row	Column	SMILES
1	1	O=C1NC(CCCF)c2[nH]nnc21
	2	OCC(OSBr)c1ccc(-c2cccc(Cl)c2Cl)[nH]1
	3	C=CC1=CC=c2c(cc(=C3ONC(Cl)=C3Cl)[nH]c2=O)O1
	4	COC(=CN1NC=CN1)C=C1C=CC(Cl)=CO1
2	1	[NH-]C(CNS1(=O)=NNc2c(F)cccc21)C1CC1
	2	CS(=O)N1CC[SH](C)C(CNCC2cccc2)C1
	3	C=C(Cl)[SH](=O)(NC)C1c2ccc(Cl)c(n2)CC1O
	4	CC(F)C(=C1C[NH2+]C([O-])N1)S(=O)Cc1cccc1
3	1	CC1(NC2=CONN2c2cccc2)C=C(C=O)N[N-]1
	2	CC(=O)NN1N=C(C(O)c2cccc3ccoc23)C(=O)C1=O
	3	C=CC(C)=C1C(F)=CC(C=C2ONN=NS2=O)=C1SCl
	4	CCN1ON=C(C=C(Cl)c2ccco2)C(F)(F)C1=O
4	1	O=C(CCN(c1[nH+]cc(S)s1)c1ccc2cc1SC2)C1=NCC=C1Cl
	2	CC=CNC1=C2Oc3cccc3C(C)S2=S=N1
	3	O=C(SC1=CC=NS1(=O)=O)c1ccc(Cl)cc1S1=NN=NN=N1
	4	COCCNCC1cc2cccc2[nH]1
5	1	ClC=C1CON=C(c2ncno2)N1CC(Cl)(Br)Br
	2	CS(=O)(=O)c1ccc[nH+]c1SNCC1cccc1Cl
	3	O=S1(=O)CNS(=O)(N(c2cccc2F)c2cccc2Cl)=N1
	4	O=C1NS(c2cccc2Cl)=S2(=NSN=N2)O1

Table 4.4: SMILES of the generated molecules included in Fig. 4.8. Online drawing tool: <https://pubchem.ncbi.nlm.nih.gov/edit3/index.html>

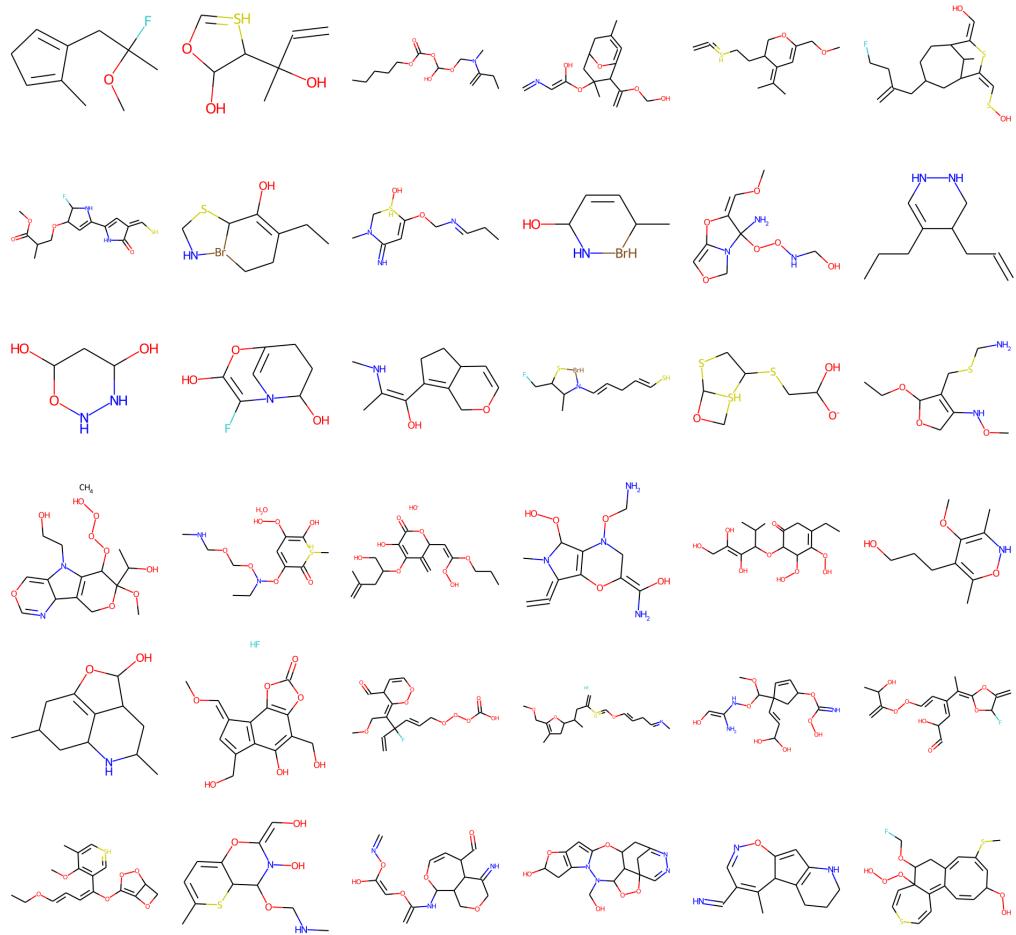


Figure 4.7: Some generated examples on ZINC by the all-at-once MGVAE with second order \mathbb{S}_n -equivariant decoders. In addition of graph features such as one-hot atomic types, we include several chemical features computed from RDKit (as in Table 4.3) as the input for the encoders. A generated example can contain more than one connected components, each of them is a valid molecule.

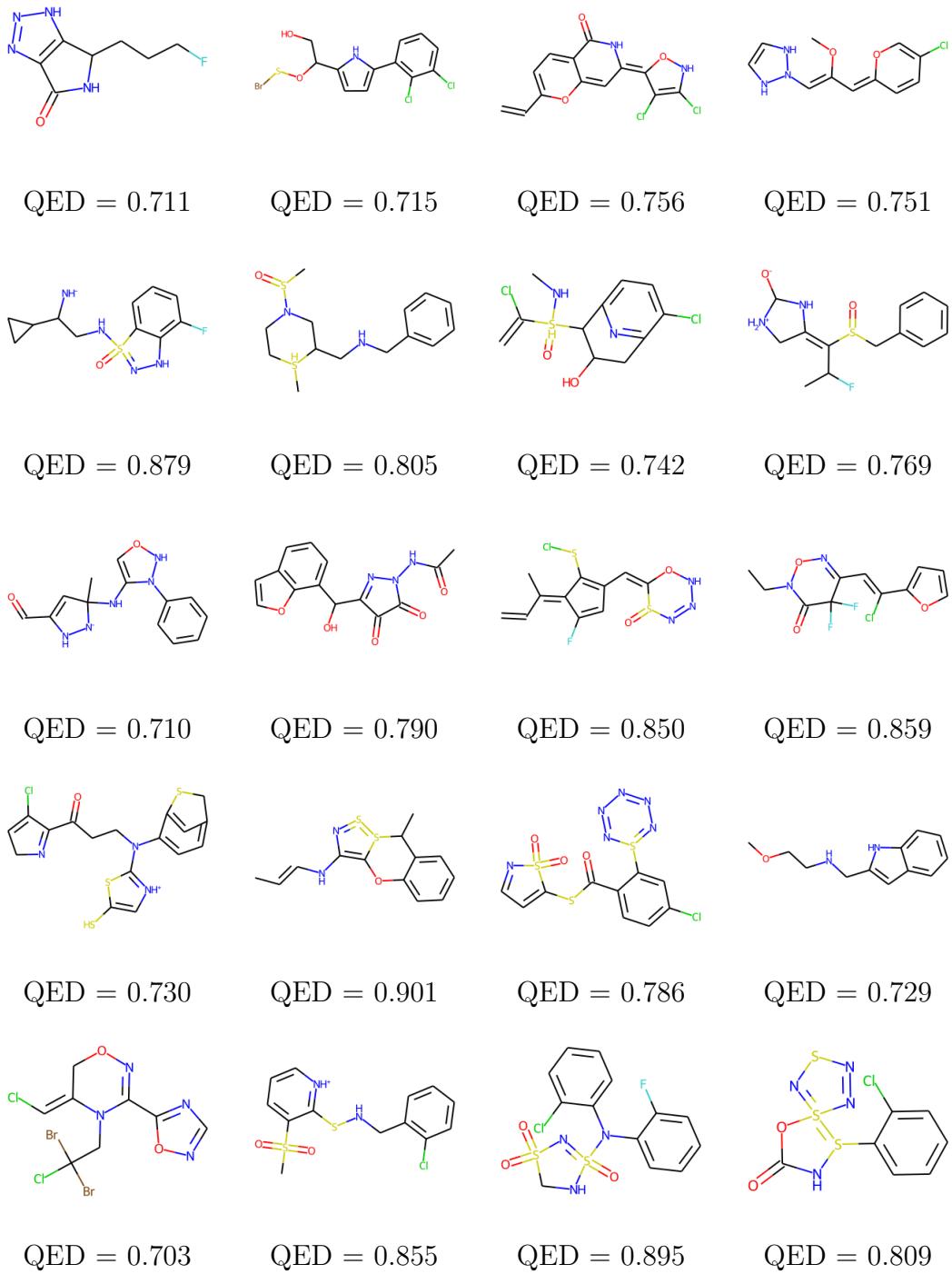


Figure 4.8: Some generated molecules on ZINC by the autoregressive MGN with high QED (drug-likeness score).

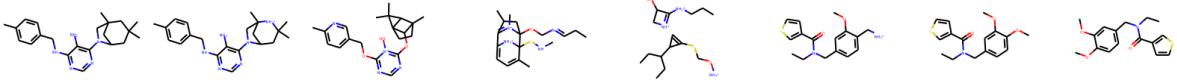


Figure 4.9: Interpolation on the latent space: we randomly select two molecules from ZINC and we reconstruct the corresponding molecular graphs on the interpolation line between the two latents.

Target	Unit	Mean	STD	Description
α	bohr ³	75.2808	8.1729	Norm of the static polarizability
C_v	cal/mol/K	31.6204	4.0674	Heat capacity at room temperature
G	eV	-70.8352	9.4975	Free energy of atomization
gap	eV	6.8583	1.2841	Difference between HOMO and LUMO
H	eV	-77.0167	10.4884	Enthalpy of atomization at room temperature
HOMO	eV	-6.5362	0.5977	Highest occupied molecular orbital
LUMO	eV	0.3220	1.2748	Lowest unoccupied molecular orbital
μ	D	2.6729	1.5034	Norm of the dipole moment
ω_1	cm ⁻¹	3504.1155	266.8982	Highest fundamental vibrational frequency
R^2	bohr ²	1189.4091	280.4725	Electronic spatial extent
U	eV	-76.5789	10.4143	Atomization energy at room temperature
U_0	eV	-76.1145	10.3229	Atomization energy at 0 K
ZPVE	eV	4.0568	0.9016	Zero point vibrational energy

Table 4.5: Description and statistics of 13 learning targets on QM9.

unsupervised and variational autoencoding manner, we extract the highest resolution latents (at $\ell = L$) and use them as the molecular representations for the downstream tasks of predicting DFT’s molecular properties on QM9 including 13 learning targets. For the training, we normalize all learning targets to have mean 0 and standard deviation 1. The name, physical unit, and statistics of these learning targets are detailed in Table 4.5.

The implementation of MGVAE is the same as detailed in Sec. 4.7.1. MGVAE is trained to reconstruct the highest resolution (input) adjacency, its coarsening adjacencies and the node atomic features. In this case, we do not use any chemical features: the node atomic

	alpha	Cv	G	gap	H	HOMO	LUMO	mu	omegal	R2	U	U0	ZPVE
WL	3.75	2.39	4.84	0.92	5.45	0.38	0.89	1.03	192	154	5.41	5.36	0.51
NGF	3.51	1.91	4.36	0.86	4.92	0.34	0.82	0.94	168	137	4.89	4.85	0.45
PSCN	1.63	1.09	3.13	0.77	3.56	0.30	0.75	0.81	152	61	3.54	3.50	0.38
CCN 2D	1.30	0.93	2.75	0.69	3.14	0.23	0.67	0.72	120	53	3.02	2.99	0.35
MGVAE	2.83	0.91	1.78	0.66	1.87	0.34	0.58	0.95	195	90	1.89	1.90	0.14

Table 4.6: Unsupervised molecular representation learning by MGVAE to predict molecular properties calculated by DFT on QM9 dataset.

features are just one-hot atomic types. After MGVAE is converged, to obtain the \mathbb{S}_n -invariant molecular representation, we average the node latents at the L -th level into a vector of size 256. Finally, we apply a simple Multilayer Perceptron with 2 hidden layers of size 512, sigmoid nonlinearity and a linear layer on top to predict the molecular properties based on the extracted molecular representation. We compare the results in Mean Average Error (MAE) in the corresponding physical units with four methods on the same split of training and testing from (Hy et al., 2018):

1. Support Vector Machine on optimal-assignment Weisfeiler-Lehman (WL) graph kernel (Shervashidze et al., 2011) (Kriege et al., 2016)
2. Neural Graph Fingerprint (NGF) (Duvenaud et al., 2015)
3. PATCHY-SAN (PSCN) (Niepert et al., 2016)
4. Second order \mathbb{S}_n -equivariant Covariant Compositional Networks (CCN 2D) (Kondor et al., 2018a) (Hy et al., 2018)

Our unsupervised results show that MGVAE is able to learn a universal molecular representation in an unsupervised manner and outperforms WL in 12, NGF in 10, PSCN in 8, and CCN 2D in 8 out of 13 learning targets, respectively (see Table 4.6). There are other recent methods in the field that use several chemical and geometric information but comparing to them would be unfair.

Method	MLP	GCN	GAT	MoNet	DiscenGCN	FactorGCN	GatedGCN _E	MGN
MAE	0.667	0.503	0.479	0.407	0.538	0.366	0.363	0.290

Table 4.7: Supervised MGN to predict solubility on ZINC dataset.

4.7.3 Supervised molecular properties prediction on ZINC

To further demonstrate the comprehensiveness of MGN, we apply our model in a supervised regression task to predict the solubility (LogP) on the ZINC dataset. We use the same split of 10K/1K/1K for training/validation/testing as in (Dwivedi et al., 2020). The implementation of MGN is almost the same as detailed in Sec. 4.7.1, except we include the latents of all resolution levels into the prediction. In particular, in each resolution level, we average all the node latents into a vector of size 256; then we concatenate all these vectors into a long vector of size $256 \times L$ and apply a linear layer for the regression task. The baseline results are taken from (Yang et al., 2020) including:

1. Multilayer Perceptron (MLP),
2. Graph Convolution Networks (GCN),
3. Graph Attention Networks (GAT) (Veličković et al., 2018),
4. MoNet (Monti et al., 2017),
5. Disentangled Graph Convolutional Networks (DisenGCN) (Ma et al., 2019),
6. Factorizable Graph Convolutional Networks (FactorGCN) (Yang et al., 2020),
7. GatedGCN_E (Dwivedi et al., 2020) that uses additional edge information.

Our supervised result shows that MGN outperforms the state-of-the-art models in the field with a margin of 20% (see Table 4.7).

4.7.4 General graph generation by MGVAE

We further examine the expressive power of hierarchical latent structure of MGVAE in the task of general graph generation. We choose two datasets from GraphRNN paper (You et al., 2018a):

1. **Community-small:** A synthetic dataset of 100 2-community graphs where $12 \leq |V| \leq 20$.
2. **Ego-small:** 200 3-hop ego networks extracted from the Citeseer network (Sen et al., 2008) where $4 \leq |V| \leq 18$.

The datasets are generated by the scripts from the GraphRNN codebase (You et al., 2018b). We keep 80% of the data for training and the rest for testing. We evaluate our generated graphs by computing Maximum Mean Discrepancy (MMD) distance between the distributions of graph statistics on the test set and the generated set as proposed by (You et al., 2018a). The graph statistics are node degrees, clustering coefficients, and orbit counts. As suggested by (Liu et al., 2019), we execute 15 runs with different random seeds, and we generate 1,024 graphs for each run, then we average the results over 15 runs. We compare MGVAE against GraphVAE (Simonovsky and Komodakis, 2018), DeepGMG (Li et al., 2018), GraphRNN (You et al., 2018a), GNF (Liu et al., 2019), and GraphAF (Shi et al., 2020). The baselines are taken from GNF paper (Liu et al., 2019) and GraphAF paper (Shi et al., 2020). In our setting of (all-at-once) MGVAE, we implement only $L = 2$ levels of resolution and $K = 2^\ell$ clusters for each level. Our encoders have 10 layers of message passing. Instead of using a high order equivariant network as the global decoder for the bottom resolution, we only implement a simple fully connected network that maps the latent $\mathcal{Z}^{(L)} \in \mathbb{R}^{|\mathcal{V}| \times d_z}$ into an adjacency matrix of size $|\mathcal{V}| \times |\mathcal{V}|$. For the ego dataset in particular, we implement the learnable equivariant prior as in Sec. 4.5 and Sec. 4.6. Table 4.8 includes our quantitative results in comparison with other methods. MGVAE outperforms all competing methods. Figs. 4.10 4.11 show some generated examples and training examples on

	COMMUNITY-SMALL			EGO-SMALL		
Model	DEGREE	CLUSTER	ORBIT	DEGREE	CLUSTER	ORBIT
GRAPHVAE	0.35	0.98	0.54	0.13	0.17	0.05
DEEPGMG	0.22	0.95	0.4	0.04	0.10	0.02
GRAPHRNN	0.08	0.12	0.04	0.09	0.22	0.003
GNF	0.20	0.20	0.11	0.03	0.10	0.001
GRAPHAF	0.06	0.10	0.015	0.04	0.04	0.008
MGVAE	0.002	0.01	0.01	1.74e-05	0.0006	6.53e-05

Table 4.8: Graph generation results depicting MMD for various graph statistics between the test set and generated graphs. MGVAE outperforms all competing methods.

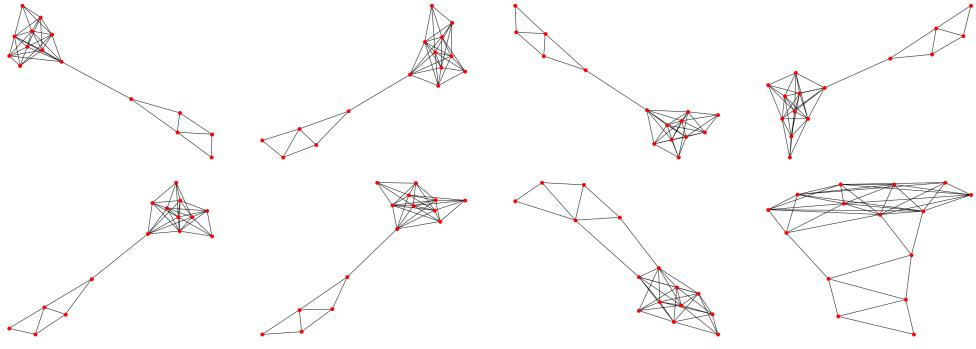
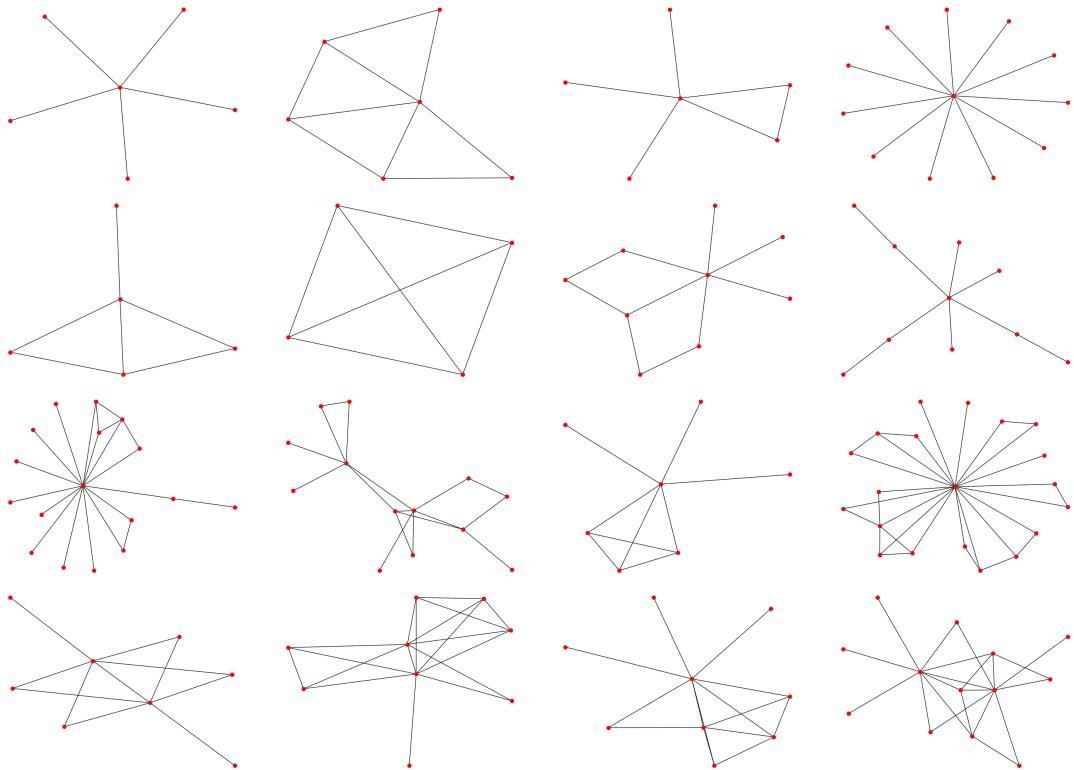


Figure 4.10: The top row includes generated examples and the bottom row includes training examples on the synthetic 2-community dataset.

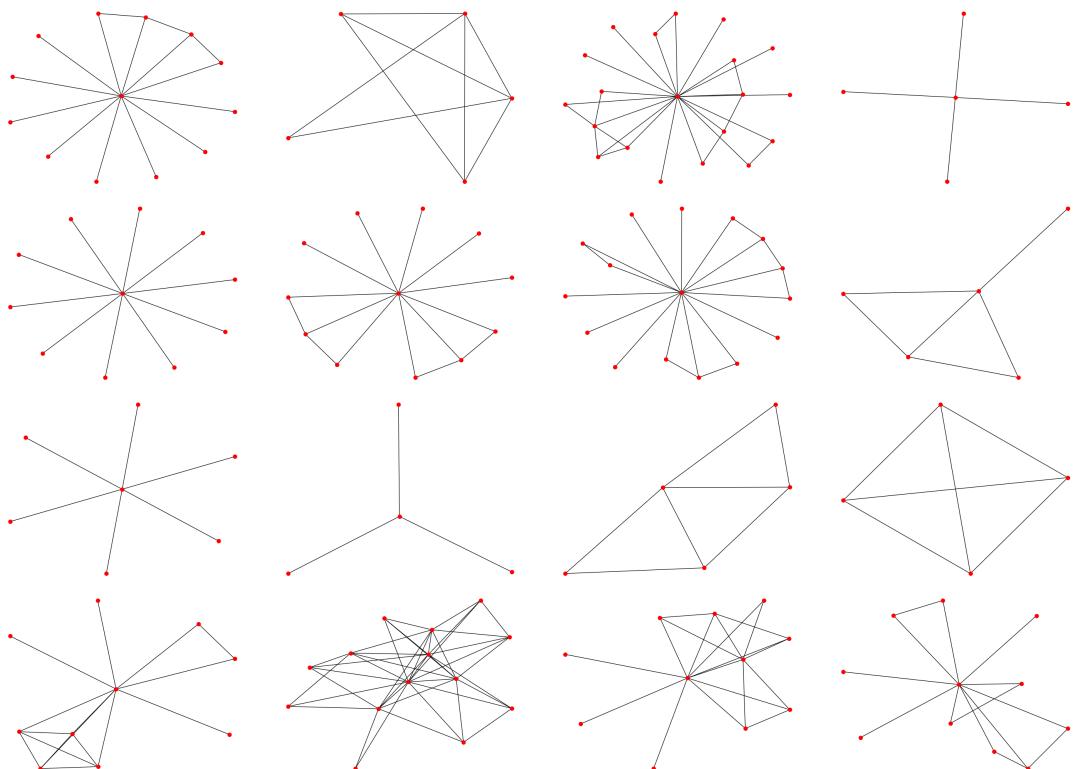
the 2-community and ego datasets.

4.7.5 Link prediction on citation graphs by MGVAE

We demonstrate the ability of the MGVAE models to learn meaningful latent embeddings on a link prediction task on popular citation network datasets Cora and Citeseer (Sen et al., 2008). At training time, 15% of the citation links (edges) were removed while all node features are kept, the models are trained on an incomplete graph Laplacian constructed from the remaining 85% of the edges. From previously removed edges, we sample the same number of pairs of unconnected nodes (non-edges). We form the validation and test sets that contain 5% and 10% of edges with an equal amount of non-edges, respectively.



Generated examples



Training examples

Figure 4.11: **EGO-SMALL**.
182

We compare our model MGVAE against popular methods in the field:

1. Spectral clustering (SC) (Tang and Liu, 2011)
2. Deep walks (DW) (Perozzi et al., 2014)
3. Variational graph autoencoder (VGAE) (Kipf and Welling, 2016)

on the ability to correctly classify edges and non-edges using two metrics: area under the ROC curve (AUC) and average precision (AP). Numerical results of SC and DW are experimental settings are taken from (Kipf and Welling, 2016). We reran the implementation of VGAE as in (Kipf and Welling, 2016).

For MGVAE, we initialize weights by Glorot initialization (Glorot and Bengio, 2010). We repeat the experiments with 5 different random seeds and calculate the average AUC and AP along with their standard deviations. The number of message passing layers ranges from 1 to 4. The size of latent representation is 128. The number of coarsening levels is $L \in \{3, 7\}$. In the ℓ -th coarsening level, we partition the graph $\mathcal{G}^{(\ell)}$ into 2^ℓ (for $L = 7$) or 4^ℓ (for $L = 3$) clusters. We train for 2,048 epochs using Adam optimization (Kingma and Ba, 2015) with a starting learning rate of 0.01. Hyperparameters optimization (e.g. number of layers, dimension of the latent representation, etc.) is done on the validation set. MGVAE outperforms all other methods (see Table 4.9).

We propose our learning to cluster algorithm to achieve the balanced K -cut at every resolution level. Besides, we also implement two fixed clustering algorithms:

1. **Spectral**: It is similar to the one implemented in (Rustamov and Guibas, 2013).

- First, we embed each node $i \in \mathcal{V}$ into $\mathbb{R}^{n_{max}}$ as $(\xi_1(i)/\lambda_1(i), \dots, \xi_{n_{max}}(i)/\lambda_{n_{max}}(i))$, where $\{\lambda_n, \xi_n\}_{n=0}^{n_{max}}$ are the eigen-pairs of the graph Laplacian $\mathcal{L} = \mathcal{D}^{-1}(\mathcal{D} - \mathcal{A})$ where $\mathcal{D}_{ii} = \sum_j \mathcal{A}_{ij}$. We assume that $\lambda_0 \leq \dots \leq \lambda_{n_{max}}$. In this case, $n_{max} = 10$.

- At the ℓ -th resolution level, we apply the K-Means clustering algorithm based on the above node embedding to partition graph $\mathcal{G}^{(\ell)}$.

2. K-Means:

- First, we apply PCA to compress the sparse word frequency vectors (of size 1,433 on Cora and 3,703 on Citeseer) associating with each node into 10 dimensions.
- We use the compressed node embedding for the K-Means clustering.

Tables 4.10 and 4.11 show that our learning to cluster algorithm returns a much more balanced cut on the highest resolution level comparing to both Spectral and K-Means clusterings. For instance, we have $L = 7$ resolution levels and we partition the ℓ -th resolution into $K = 2^\ell$ clusters. Thus, on the bottom levels, we have 128 clusters. If we distribute nodes into clusters uniformly, the expected number of nodes in a cluster is 21.15 and 25.99 on Cora (2,708 nodes) and Citeseer (3,327 nodes), respectively. We measure the minimum, maximum, standard deviation of the numbers of nodes in 128 clusters. Furthermore, we measure the Kullback–Leibler divergence between the distribution of nodes into clusters and the uniform distribution. Our learning to cluster algorithm achieves low KL losses of 0.02 and 0.01 on Cora and Citeseer, respectively.

Table 4.9: Citation graph link prediction results (AUC & AP)

Dataset	Cora		Citeseer	
Method	AUC (ROC)	AP	AUC (ROC)	AP
SC	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01
DW	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01
VGAE	90.97 ± 0.77	91.88 ± 0.83	89.63 ± 1.04	91.10 ± 1.02
MGVAE (Spectral)	91.19 ± 0.76	92.27 ± 0.73	90.55 ± 1.17	91.89 ± 1.27
MGVAE (K-Means)	93.07 ± 5.61	92.49 ± 5.77	90.81 ± 1.19	91.98 ± 1.02
MGVAE	95.67 ± 3.11	95.02 ± 3.36	93.93 ± 5.87	93.06 ± 6.33

Method	Min	Max	STD	KL divergence
Spectral	1	2020	177.52	3.14
K-Means	1	364	40.17	0.84
Learn to cluster	10	36	4.77	0.02

Table 4.10: Learning to cluster algorithm returns balanced cuts on Cora.

Method	Min	Max	STD	KL divergence
Spectral	1	3320	292.21	4.51
K-Means	1	326	41.69	0.74
Learn to cluster	11	38	4.93	0.01

Table 4.11: Learning to cluster algorithm returns balanced cuts on Citeseer.

4.7.6 Graph-based image generation by MGVAE

In this additional experiment, we apply MGVAE into the task of image generation. Instead of matrix representation, an image $I \in \mathbb{R}^{H \times W}$ is represented by a grid graph of $H \cdot W$ nodes in which each node represents a pixel, each edge is between two neighboring pixels, and each node feature is the corresponding pixel's color (e.g., \mathbb{R}^1 in gray scale, and \mathbb{R}^3 in RGB scale). Fig. 4.12 demonstrates an example of graph representation for images. Since images have natural spatial clustering, instead of learning to cluster, we implement a fixed clustering procedure as follows:

- For the ℓ -th resolution level, we divide the grid graph of size $H^{(\ell)} \times W^{(\ell)}$ into clusters of size $h \times w$ that results into a grid graph of size $\frac{H^{(\ell)}}{h} \times \frac{W^{(\ell)}}{w}$, supposedly h and w are divisible by $H^{(\ell)}$ and $W^{(\ell)}$, respectively. Each resolution is associated with an image $I^{(\ell)}$ that is a zoomed out version of $I^{(\ell+1)}$.
- The global encoder $e^{(\ell)}$ is implemented with 10 layers of message passing that operates on the whole $H^{(\ell)} \times W^{(\ell)}$ grid graph. We sum up all the node latents into a single latent vector $Z^{(\ell)} \in \mathbb{R}^{d_z}$. The global decoder $d^{(\ell)}$ is implemented by the convolutional neural network architecture of the generator of DCGAN model (Radford et al., 2016) to

map $Z^{(\ell)}$ into an approximated image $\hat{I}^{(\ell)}$. The \mathbb{S}_n -invariant pooler $\mathbf{p}^{(\ell)}$ is a network operating on each small $h \times w$ grid graph to produce the corresponding node feature for the next level $\ell + 1$. MGVAE is trained to reconstruct all resolution images. Fig. 4.13 shows an example of reconstruction at each resolution on a test image of MNIST (after the network converged).

We evaluate our MGVAE architecture on the MNIST dataset (LeCun et al., 1999) with 60,000 training examples and 10,000 testing examples. The original image size is 28×28 . We pad zero pixels to get the image size of $2^5 \times 2^5$ (e.g., $H^{(5)} = W^{(5)} = 32$). Each cluster is a small grid graph of size 2×2 (e.g., $h = w = 2$). Accordingly, the image sizes for all resolutions are 32×32 , 16×16 , 8×8 , etc. In this case, the whole network architecture is a 2-dimensional quadtree. The latent size d_z is selected as 256. We train our model for 256 epochs by Adam optimizer (Kingma and Ba, 2015) with the initial learning rate 10^{-3} . In the testing process, for the ℓ -th resolution, we sample a random vector of size d_z from prior $\mathcal{N}(0, 1)$ and use the decoder $\mathbf{d}^{(\ell)}$ to decode the corresponding image. We generate 10,000 examples for each resolution. We compute the Frechet Inception Distance (FID) proposed by (Heusel et al., 2017) between the testing set and the generated set as the metric to evaluate the quality of our generated examples. We use the FID implementation from (Seitzer, 2020). We compare our MGVAE against variants of Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) including DCGAN (Radford et al., 2016), VEEGAN (Srivastava et al., 2017), PacGAN (Lin et al., 2018), and PresGAN (Dieng et al., 2019). Table 4.12 shows our quantitative results in comparison with other competing generative models. The baseline results are taken from *Prescribed Generative Adversarial Networks* paper (Dieng et al., 2019). MGVAE outperforms all the baselines for the highest resolution generation. Figs. 4.14 and 4.15 show some generated examples of the 32×32 and 16×16 resolution, respectively.

Method	$\text{FID}_{\downarrow} (32 \times 32)$	$\text{FID}_{\downarrow} (16 \times 16)$	$\text{FID}_{\downarrow} (8 \times 8)$
DCGAN	113.129		
VEEGAN	68.749	N/A	N/A
PACGAN	58.535		
PresGAN	42.019		
MGVAE	39.474	64.289	39.038

Table 4.12: Quantitative evaluation of the generated set by FID metric for each resolution level on MNIST. It is important to note that the generation for each resolution is done separately: for the ℓ -th resolution, we sample a random vector of size $d_z = 256$ from $\mathcal{N}(0, 1)$, and use the global decoder $\mathbf{d}^{(\ell)}$ to decode into the corresponding image size. The baselines are taken from (Dieng et al., 2019).

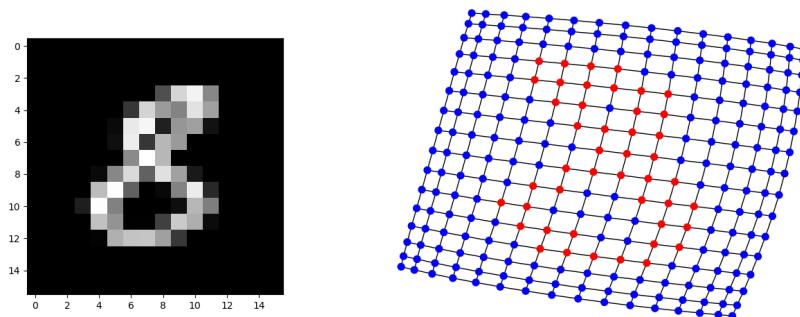


Figure 4.12: An image of digit 8 from MNIST (left) and its grid graph representation at 16×16 resolution level (right).

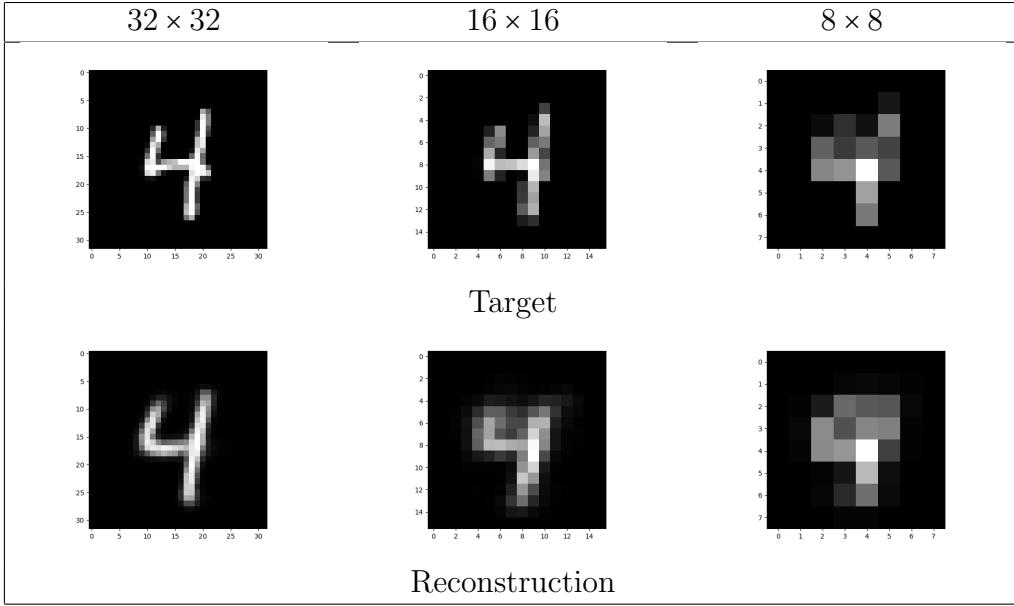


Figure 4.13: An example of reconstruction on each resolution level for a test image in MNIST.

4.8 Software

Our Pythonic implementation of MGVAE and MGN in PyTorch is available at:

<https://github.com/HyTruongSon/MGVAE>

4.9 Chapter Conclusion

In this chapter, we introduced MGVAE built upon MGN, the first generative model to learn and generate graphs in a multiresolution and equivariant manner. The key idea of MGVAE is learning to construct a series of coarsened graphs along with a hierarchy of latent distributions in the encoding process while learning to decode each latent into the corresponding coarsened graph at every resolution level. MGVAE achieves state-of-the-art results from link prediction to molecule and graph generation, suggesting that accounting for the multiscale structure of graphs is a promising way to make graph neural networks even more powerful.



Figure 4.14: Generated examples at the highest 32×32 resolution level.

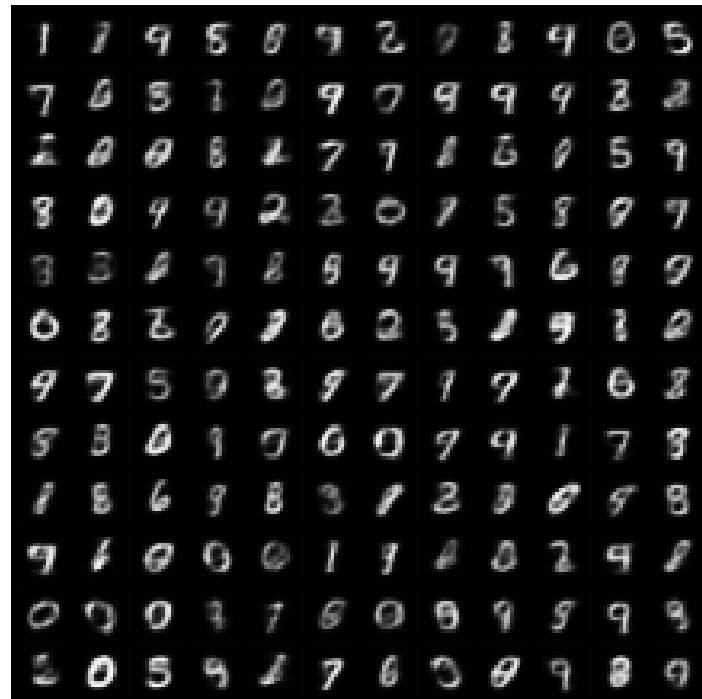


Figure 4.15: Generated examples at the 16×16 resolution level.

CHAPTER 5

ROTATIONALLY EQUIVARIANT MOLECULAR NEURAL NETWORKS

5.1 Chapter Introduction

In general, we want to learn on molecular data specified by a set of charge-position pairs (Z_i, \mathbf{r}_i) for each atom (see figure 5.1). This problem is invariant to rotations and translations. We use covariant activations to “bake-in” these symmetries, while retaining local geometric information. In this chapter, we introduce *Cormorant* (Anderson et al., 2019), a rotationally covariant neural network architecture for learning the behavior and properties of complex many-body physical systems. We apply these networks to molecular systems with two goals: learning atomic potential energy surfaces for use in Molecular Dynamics simulations, and learning ground state properties of molecules calculated by Density Functional Theory. Some of the key features of our network are that (a) each neuron explicitly corresponds to a subset of atoms; (b) the activation of each neuron is covariant to rotations, ensuring that overall the network is fully rotationally invariant. Furthermore, the non-linearity in our network is based upon tensor products and the Clebsch-Gordan decomposition, allowing the network to operate entirely in Fourier space. *Cormorant* significantly outperforms competing algorithms in learning molecular Potential Energy Surfaces from conformational geometries in the MD-17 dataset, and is competitive with other methods at learning geometric, energetic, electronic, and thermodynamic properties of molecules on the GDB-9 dataset.

5.2 Related work

In principle, quantum mechanics provides a perfect description of the forces governing the behavior of atoms, molecules and crystalline materials such as metals. However, for systems

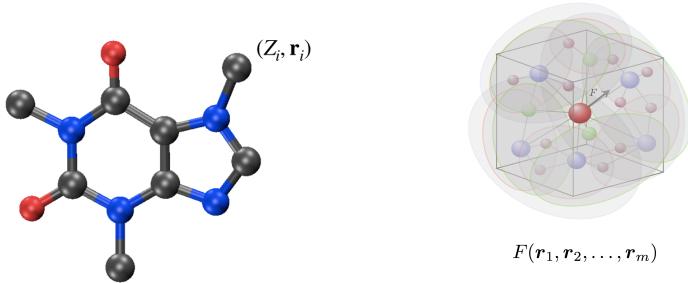


Figure 5.1: Learning on molecular data specified by a set of charge-position pairs (Z_i, \mathbf{r}_i) for each atom.

larger than a few dozen atoms, solving the Schrödinger equation explicitly at every timestep is not a feasible proposition on present day computers. Even Density Functional Theory (DFT) (Hohenberg and Kohn, 1964), a widely used approximation to the equations of quantum mechanics, has trouble scaling to more than a few hundred atoms.

Consequently, the majority of practical work in molecular dynamics today falls back on fundamentally classical models, where the atoms are essentially treated as solid balls and the forces between them are given by pre-defined formulae called *atomic force fields* or *empirical potentials*, such as the CHARMM family of models (Brooks et al., 1983, 2009). There has been a widespread realization that this approach has inherent limitations, so in recent years a burgeoning community has formed around trying to use machine learning to *learn* more descriptive force fields directly from DFT computations (Behler and Parrinello, 2007; Bartók et al., 2010; Rupp et al., 2012; Shapeev, 2015; Chmiela et al., 2016; Zhang et al., 2017; Schütt et al., 2017; Hirn et al., 2017). More broadly, there is considerable interest in using ML methods not just for learning force fields, but also for predicting many other physical/chemical properties of atomic systems across different branches of materials science, chemistry and pharmacology (Montavon et al., 2013; Gilmer et al., 2017; Smith et al., 2017; Yao et al., 2018).

At the same time, there have been significant advances in our understanding of the equivari-

ance and covariance properties of neural networks, starting with (Cohen and Welling, 2016, 2017) in the context of traditional convolutional neural nets (CNNs). Similar ideas underly generalizations of CNNs to manifolds (Masci et al., 2015; Monti et al., 2017; Bronstein et al., 2017) and graphs (Bruna et al., 2014; Henaff et al., 2015). In the context of CNNs on the sphere, (Cohen et al., 2018) realized the advantage of using “Fourier space” activations, i.e., expressing the activations of neurons in a basis defined by the irreducible representations of the underlying symmetry group (see also (Esteves et al., 2017)), and these ideas were later generalized to the entire SE(3) group (Weiler et al., 2018). (Kondor and Trivedi, 2018) gave a complete characterization of what operations are allowable in Fourier space neural networks to preserve covariance, and Cohen et al generalized the framework even further to arbitrary gauge fields (Cohen et al., 2019). There have also been some recent works where even the nonlinear part of the neural network’s operation is performed in Fourier space: independently of each other (Thomas et al., 2018) and (Kondor, 2018) were to first to use the Clebsch–Gordan transform inside rotationally covariant neural networks for learning physical systems, while (Kondor et al., 2018b) showed that in spherical CNNs the Clebsch–Gordan transform is sufficient to serve as the sole source of nonlinearity.

5.3 Motivation from physics

5.3.1 *The multipole expansion*

From a chemical point of view, effective atom-atom interactions break down into a few simple classes based on symmetry. Here we review a few of these classes in the context of the multipole expansion, whose structure will inform the design of our molecular neural networks:

- **Scalar interactions:** Electrostatic attraction/repulsion between two charges described

by the Coulomb energy

$$V_C = -\frac{1}{4\pi\epsilon_0} \frac{q_A q_B}{r_{AB}}$$

where q_A and q_B are the charges of the two particles, \mathbf{r}_A and \mathbf{r}_B are the position vectors, $r_{AB} = |\mathbf{r}_{AB}| = |\mathbf{r}_A - \mathbf{r}_B|$, and ϵ_0 is a universal constant.

- **Dipole/dipole interactions:** The electrostatic dipole moment of a set of N charged particles relative to their center of mass \mathbf{r} is the first moment of their position vectors weighted by their charges

$$\boldsymbol{\mu} = \sum_{i=1}^N q_i (\mathbf{r}_i - \mathbf{r}).$$

The dipole/dipole contribution to the electrostatic potential energy between two sets of particles A and B separated by a vector \mathbf{r}_{AB} is given by

$$V_{d/d} = \frac{1}{4\pi\epsilon_0} \left[\frac{\boldsymbol{\mu}_A \cdot \boldsymbol{\mu}_B}{|\mathbf{r}_{AB}|^3} - 3 \frac{(\boldsymbol{\mu}_A \cdot \mathbf{r}_{AB})(\boldsymbol{\mu}_B \cdot \mathbf{r}_{AB})}{|\mathbf{r}_{AB}|^5} \right].$$

- **Quadropole/quadropole interactions:** In the electrostatic case, the quadropole moment is the second moment of the charge density, described by the matrix

$$\boldsymbol{\Theta} = \sum_{i=1}^N q_i (3\mathbf{r}_i \mathbf{r}_i^T - |\mathbf{r}_i|^2 I).$$

Higher order interactions involve moment tensors of order 3, 4, 5, and so on.

The construction of our architecture takes the inspiration from the multipole expansion in physics. Expanding the potential $V(\mathbf{r})$ around a point \mathbf{r} gives (see Fig. 5.2):

$$\sum_i \frac{Z_i}{|\mathbf{r} - \mathbf{r}_i|} = Q_0 Y^0(\hat{\mathbf{r}})/r + Q_1 Y^1(\hat{\mathbf{r}})/r^2 + Q_2 Y^2(\hat{\mathbf{r}})/r^3 + \dots$$

Here Q_ℓ is the ℓ -th multipole moment, and $Y^\ell(\hat{\mathbf{r}})$ is a spherical harmonic. Figure 5.3 shows the effects of a 90° CCW-rotation on the input charges and corresponding moments. All

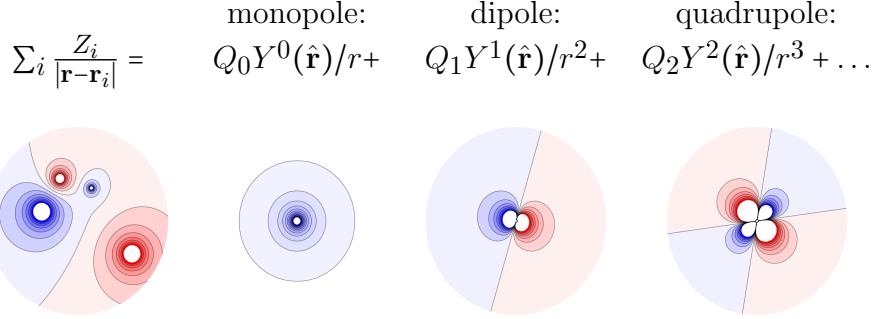


Figure 5.2: Potential expansion.

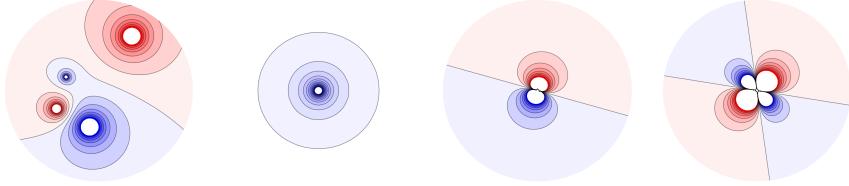


Figure 5.3: Effects of a 90° CCW-rotation on the input charges and corresponding moments.

moments rotate **covariantly**. More precisely, given a rotation $R \in \text{SO}(3)$, each moment rotates as $Q_\ell \rightarrow D^\ell(R) \cdot Q_\ell$, where $D^\ell : \text{SO}(3) \rightarrow \mathbb{C}^{(2\ell+1) \times (2\ell+1)}$ is a Wigner- D matrix or irreducible representation (irrep) of the group $\text{SO}(3)$ labeled by positive integer $\ell = 0, 1, 2, \dots$.

5.3.2 Spherical tensors

Physical quantities transform under rotation $R \in \text{SO}(3)$:

$$q \mapsto q \quad \boldsymbol{\mu} \mapsto R\boldsymbol{\mu} \quad \boldsymbol{\Theta} \mapsto R\boldsymbol{\Theta}R^T \quad \mathbf{r}_{AB} \mapsto R\mathbf{r}_{AB}$$

In general, a k 'th order Cartesian moment tensor $T^{(k)} \in \mathbb{R}^{3 \times 3 \times \dots \times 3}$ (or its flattened $\overline{T}^{(k)} \in \mathbb{R}^{3k}$) transforms as:

$$\overline{T}^{(k)} \mapsto (R \otimes R \otimes \dots \otimes R) \overline{T}^{(k)}$$

Recall that given a group G , a *representation* ρ of G is a matrix valued function $\rho : G \rightarrow \mathbb{C}^{d \times d}$ obeying $\rho(xy) = \rho(x)\rho(y)$ for any two group elements $x, y \in G$. It is easy to see that R , and

consequently $R \otimes R \otimes \dots \otimes R$ are the representations of the three dimensional rotation group $\text{SO}(3)$. In the specific case of $\text{SO}(3)$, there is a fixed unitary transformation matrix $C^{(k)}$ which reduces the k 'th order rotation operator into a direct sum of irreducible representations (irreps):

$$\underbrace{R \otimes R \otimes \dots \otimes R}_k = C^{(k)} \left[\bigoplus_{\ell} \bigoplus_{i=1}^{\tau_{\ell}} D^{\ell}(R) \right] C^{(k)\dagger}$$

where the irreps $D^{\ell} : \text{SO}(3) \rightarrow \mathbb{C}^{(2\ell+1) \times (2\ell+1)}$ are called Wigner D -matrices for any positive integer $\ell = 0, 1, 2, \dots$. The vectorized form of the Cartesian moment tensor $\bar{T}^{(k)}$ has a corresponding decomposition:

$$\bar{T}^{(k)} = C^{(k)} \left[\bigoplus_{\ell} \bigoplus_{i=1}^{\tau_{\ell}} Q_{\ell,i} \right]$$

Under rotations, the individual $Q_{\ell,i}$ components transform independently as $Q_{\ell,i} \mapsto D^{\ell}(R)Q_{\ell,i}$. For example, in the dipole/dipole case, we need terms of the form $Q_{\ell_1}^A \otimes Q_{\ell_2}^B$ that transform according to the tensor product of the corresponding irreps:

$$Q_{\ell_1}^A \otimes Q_{\ell_2}^B \mapsto (D^{\ell_1}(R) \otimes D^{\ell_2}(R))(Q_{\ell_1}^A \otimes Q_{\ell_2}^B)$$

in which $D^{\ell_1}(R) \otimes D^{\ell_2}(R)$ is not an irreducible representation, but can be decomposed into irreducibles by **Clebsch - Gordan** decomposition:

$$D^{\ell_1}(R) \otimes D^{\ell_2}(R) = C_{\ell_1, \ell_2}^{\dagger} \left[\bigoplus_{\ell=|\ell_1-\ell_2|}^{\ell_1+\ell_2} D^{\ell}(R) \right] C_{\ell_1, \ell_2}.$$

5.4 Molecular neural networks based on the Clebsch-Gordan layer

Definition 5.4.1 (SO(3)-covariant vector). We say that F is an SO(3)-covariant vector of type $\tau = (\tau_0, \tau_1, \tau_2, \dots, \tau_L)$ if it can be written as a collection of complex matrices F_0, F_1, \dots, F_L , called its isotypic parts, where each F_ℓ is a matrix of size $(2\ell + 1) \times \tau_\ell$ and transforms under rotations as $F_\ell \mapsto D^\ell(R)F_\ell$.

Definition 5.4.2 (Molecular neural networks). Let S be a molecule or other physical system consisting of N atoms. A **Cormorant** covariant molecular neural network for S is a feed forward neural network consisting of m neurons $\mathbf{n}_1, \dots, \mathbf{n}_m$ such that:

- Every neuron \mathbf{n}_i corresponds to some subset S_i of the atoms. In particular, each input neuron corresponds to a single atom. Each output neuron corresponds to the entire system S .
- The activation of each \mathbf{n}_i is an SO(3) - vector of a fixed type τ_i .
- The type of each output neuron is $\tau_{out} = (1)$, i.e., a scalar.

Cormorant is based upon the Clebsch-Gordan layer. We use covariant SO(3)-vectors $F_i^s = \bigoplus_{\ell=0}^L \bigoplus_c [F_\ell]_{c,i}^s$ as activations for each atom i at layer s . The Clebsch-Gordan decomposition is the central operation in our network:

$$[[F_{\ell_1} \otimes_{cg} G_{\ell_2}]_\ell]_{*,i} = \bigoplus_{\ell=|\ell_1-\ell_2|}^{\ell_1+\ell_2} C_{\ell_1, \ell_2, \ell} ([F_{\ell_1}]_{*,i} \otimes [G_{\ell_2}]_{*,i}).$$

Using this operation, we update our activations in a two-step process. Atom activations are updated using:

$$F_i^s = \underbrace{\left[F_i^{s-1} \oplus (F_i^{s-1} \otimes_{cg} F_i^{s-1}) \oplus \left(\sum_j G_{i,j}^s \otimes_{cg} F_j^{s-1} \right) \right]}_{\text{one-body part}} \cdot \underbrace{W_{s,\ell}^{\text{vertex}}}_{\text{two-body part}},$$

where $W_{s,\ell}^{\text{vertex}}$ is a set of learnable weights. Note that the form of the CG non-linearity is enforced by group theory. Any other covariant non-linearity must be either composed of CG operations, or scalars.

The SO(3)-vector edge activations $G_{i,j}^{s,\ell} = g_{i,j}^{s,\ell} Y^\ell(\hat{\mathbf{r}}_{ij})$ are constructed from the spherical harmonics $Y^\ell(\hat{\mathbf{r}}_{ij})$ of the relative position vector $\mathbf{r}_{i,j} = \mathbf{r}_i - \mathbf{r}_j$ between atoms i and j , along with the scalar edge network $g_{i,j}^{s,\ell}$. Specifically,

$$g_{i,j}^{s,\ell} = \mu^s(r_{i,j}) \left[\left(g_{i,j}^{s-1,\ell} \oplus (F_i^{s-1} \cdot F_j^{s-1}) \oplus \eta^{s,\ell}(r_{i,j}) \right) \cdot W_{s,\ell}^{\text{edge}} \right],$$

where $\mu^s(r_{i,j})$ is a learnable mask, $\eta^{s,\ell}(r_{i,j})$ is a set of learnable radial basis functions of the relative distance $r_{i,j}$, and $W_{s,\ell}^{\text{edge}}$ are learnable weights.

5.5 Experiments

We present experimental results on two datasets of interest to the computational chemistry community: MD-17 for learning molecular force fields and potential energy surfaces, and QM-9 for learning the ground state properties of a set of molecules.

For both datasets, the central covariant SO(3)-vector layers of our Cormorant are identical. In both cases, we used $S = 4$ layers with $L = 3$, followed by a single SO(3)-vector layer with $L = 0$. The number of channels of the input tensors at each level is fixed to $n_c = 16$, and similarly the set of weights W reduce the number of channels of each irreducible representation back to $n_c = 16$.

We trained our network using the AMSGrad (Reddi et al., 2018) optimizer with a constant learning rate of 5×10^{-4} and a mini-batch size of 25. We trained for 512 and 256 epoch

respectively for MD-17 and QM-9.

5.5.1 Learning the ground state properties

QM9 (Ramakrishnan et al., 2014) is a dataset of approximately 134k small organic molecules containing the atoms H, C, N, O, F. For each molecule, the ground state configuration is calculated using DFT, along with a variety of molecular properties. We use the ground state configuration as the input to our Cormorant, and use a common subset of properties in the literature as regression targets.

We used an input featurization based upon message passing neural networks. We start by creating the vector $\tilde{F}_i = \text{onehot}_i \otimes \vec{Z}_i$ as defined in the previous section. Using this, a weighted adjacency matrix is constructed using a mask in the same manner as in the main text: $\mu_{ij} = \sigma((r_{\text{cut}} - r_{ij})/w)$, with learnable cutoffs/width r_{cut}/w and $\sigma(x) = 1/(1 + \exp(-x))$. This mask is used to aggregate neighbors $\tilde{F}_i^{\text{agg}} = \sum_j \mu_{ij} \tilde{F}_j$. The result is concatenated with \tilde{F}_i , and passed through a MLP with a single hidden layer with 256 neurons and ReLU activations with an output real vector of length $2 \times n_c$. This is then resized to form a complex SO(3)-vector composed of a single irrep of type $\tau_i = (n_c)$.

The output for the QM-9 is constructed using two multi-layer perceptrons (MLPs). First, a MLP is applied to the scalar representation x_i at each site. The result is summed over all sites, forming a single permutation invariant representation of the molecule. This representation is then used to predict a single number used as the regression target: $y = \text{MLP}_2(\sum_i \text{MLP}_1(x_i))$. Here, both MLP_1 and MLP_2 have a single hidden layer of size 256, and the intermediate representation has 96 neurons.

Table 5.1 presents our results compared with SchNet (Schütt et al., 2017), MPNNs (Gilmer

Table 5.1: Mean absolute error of various prediction targets on QM-9. Results within 5% of the best model are indicated in bold.

	Cormorant	SchNet	NMP	WaveScatt
α (bohr ³)	0.092	0.235	0.092	0.160
$\Delta\epsilon$ (eV)	0.060	0.063	0.069	0.118
ϵ_{HOMO} (eV)	0.036	0.041	0.043	0.085
ϵ_{LUMO} (eV)	0.036	0.034	0.038	0.076
μ (D)	0.130	0.033	0.030	0.340
C_v (cal/mol K)	0.031	0.033	0.040	0.049
R^2 (bohr ²)	0.673	0.073	0.180	0.410
U_0 (eV)	0.028	0.014	0.020	0.022
ZPVE (meV)	1.982	1.700	1.500	2.000

et al., 2017), and wavelet scattering networks (Hirn et al., 2017). Of the nine regression targets considered, we achieve leading or competitive results on five (α , C_v , $\Delta\epsilon$, ϵ_{HOMO} , ϵ_{LUMO}). These targets are all close enough with the competing architectures are to be indistinguishable. The remaining four targets are within a factor of two of the best result, with the exception of R^2 , which is much larger than the competitors.

5.5.2 Learning potential energy surfaces

MD-17 (Chmiela et al., 2016) is a dataset of eight small organic molecules (see Table 5.2) containing up to 17 total atoms composed of the atoms H, C, N, O, F. For each molecule, an *ab initio* molecular dynamics simulation was run using DFT to calculate the ground state energy and forces. At intermittent timesteps, the energy, forces, and configuration (positions of each atom) were recorded. For each molecule we use a train/validation/test split of 50k/10k/10k atoms respectively.

For MD-17, the input featurization was determined by taking the tensor product $\tilde{F}_i = \text{onehot}_i \otimes \tilde{Z}_i$, where onehot_i is a one-hot vector determining which of N_{species} atomic species an atom is, and $\tilde{Z}_i = (1, \tilde{Z}_i, \tilde{Z}_i^2)$, where $\tilde{Z}_i = Z_i/Z_{\max}$, and Z_{\max} is the largest charge in

Table 5.2: Mean absolute error of conformational energies (in units of kcal/mol) on MD-17. Results within 5% of the best model are indicated in bold.

	Cormorant	DeepMD	DTNN	SchNet	GDML
Aspirin	0.103	0.201	–	0.120	0.270
Benzene	0.035	0.065	0.040	0.070	0.070
Ethanol	0.029	0.055	–	0.050	0.150
Malonaldehyde	0.056	0.092	0.190	0.080	0.160
Naphthalene	0.043	0.095	–	0.110	0.120
Salicylic Acid	0.072	0.106	0.410	0.100	0.120
Toluene	0.042	0.085	0.180	0.090	0.120
Uracil	0.045	0.085	–	0.100	0.110

the dataset. We then use a single learnable mixing matrix to convert this real vector with $3 \times N_{\text{species}}$ elements to a complex representation $\ell = 0$ and N_c channels (or $\tau_i = (n_c)$). We found for MD-17, a complex input featurization network was not significantly beneficial, and that this input parametrization was sufficiently expressive.

The output for the MD-17 network is straightforward. The scalars x_i are summed over, and then a single linear layer is applied: $y = A(\sum_i x_i) + b$.

The results of these experiments are presented in Table 5.2, where the mean-average error (MAE) is plotted on the test set for each of molecules. (All units are in kcal/mol, as consistent with the dataset and the literature.) To the best of our knowledge, the current state-of-the art algorithms on this dataset are DeepMD (Zhang et al., 2017), DTNN (Schütt et al., 2017), SchNet (Schütt et al., 2017) and GDML (Chmiela et al., 2016). Since training and testing set sizes were not consistent, we used a training set of 50k molecules to compare with all neural network based approaches. As can be seen from the table, our *Cormorant* network outperforms all competitors.

5.6 Software

The source code in PyTorch of Cormorant is publicly available at

[https://github.com/risilab/cormorant.](https://github.com/risilab/cormorant)

5.7 Chapter Conclusion

The *Cormorant* neural network architecture proposed in this chapter combines some of the insights gained from the various force field and potential learning efforts with the emerging theory of Fourier space covariant/equivariant neural networks. The important point that we stress in the following pages is that by setting up the network in such a way that each neuron corresponds to an actual set of physical atoms, and that each activation is covariant to symmetries (rotation and translation), we get a network in which the “laws” that individual neurons learn resemble known physical interactions. In addition, amongst algorithms for learning molecular properties that fully respect invariances, *Cormorant* is arguably one of the most general. Our experiments show that this generality pays off in terms of performance on standard benchmark datasets.

CHAPTER 6

LEARNING MULTIRESOLUTION MATRIX FACTORIZATION AND ITS WAVELET NETWORKS ON GRAPHS

6.1 Chapter Introduction

In certain machine learning problems large matrices have complex hierarchical structures that traditional linear algebra methods based on the low rank assumption struggle to capture. Multiresolution matrix factorization (MMF) is a relatively little used alternative paradigm that is designed to capture structure at multiple different scales. MMF has been found to be particularly effective at compressing the adjacency or Laplacian matrices of graphs with complicated structure, such as social networks (Kondor et al., 2014).

MMF factorizations have a number of advantages, including the fact that they are easy to invert and have an interpretation as a form of wavelet analysis on the matrix and consequently on the underlying graph. The wavelets can be used e.g., for finding sparse approximations of graph signals. Finding the actual MMF factorization however is a hard optimization problem combining elements of continuous and combinatorial optimization. Most of the existing MMF algorithms just tackle this with a variety of greedy heuristics and are consequently brittle: the resulting factorizations typically have large variance and most of the time yield factorizations that are far from the optimal (Teneva et al., 2016; Ithapu et al., 2017; Ding et al., 2017).

The present paper proposes an alternative paradigm to MMF optimization based on ideas from deep learning. Specifically, we employ an iterative approach to optimizing the factorization based on backpropagating the factorization error and a reinforcement learning strategy for solving the combinatorial part of the problem. While more expensive than the

greedy approaches, we find that the resulting “learnable” MMF produces much better quality factorizations and a wavelet basis that is smoother and better reflects the structure of the underlying matrix or graph. Unsurprisingly, this also means that the factorization performs better in downstream tasks.

To apply our learnable MMF algorithm to standard benchmark tasks, we also propose a wavelet extension of the Spectral Graph Networks algorithm of (Bruna et al., 2014) which we call the Wavelet Neural Network (WNN). Our experiments show that the combination of learnable MMF optimization with WNNs achieves state of the art results on several graph learning tasks. Beyond just benchmark performance, the greatly improved stability of MMF optimization process and the similarity of the hierarchical structure of the factorization to the architecture of deep neural networks opens up the possibility of MMF being tightly integrated with other learning algorithms in the future.

6.2 Related work

Compressing and estimating large matrices has been extensively studied from various directions, including (Drineas et al., 2006), (Halko et al., 2011), (Williams and Seeger, 2001) (Kumar et al., 2012), (Mahoney, 2011), (Jenatton et al., 2010). Many of these methods come with explicit guarantees but typically make the assumption that the matrix to be approximated is low rank.

MMF is more closely related to other works on constructing wavelet bases on discrete spaces, including wavelets defined based on diagonalizing the diffusion operator or the normalized graph Laplacian (Coifman and Maggioni, 2006) (Hammond et al., 2011) and multiresolution on trees (Gavish et al., 2010) (Lee et al., 2008). MMF has been used for matrix compression (Teneva et al., 2016), kernel approximation (Ding et al., 2017) and inferring semantic

relationships in medical imaging data (Ithapu et al., 2017).

Most of the combinatorial optimization problems over graphs are NP-Hard, which means that no polynomial time solution can be developed for them. Many traditional algorithms for solving such problems involve using suboptimal heuristics designed by domain experts, and only produce approximations that are guaranteed to be some factor worse than the true optimal solution. Reinforcement learning (RL) proposes an alternative to replace these heuristics and approximation algorithms by training an agent in a supervised or self-supervised manner (Bello et al., 2016) (Mazyavkina et al., 2021). (Khalil et al., 2017) proposed the use of graph embedding network as the agent to capture the current state of the solution and determine the next action. Similarly, our learning algorithm addresses the combinatorial part of the MMF problem by gradient-policy algorithm that trains graph neural networks as the RL agent.

Graph neural networks (GNNs) utilizing the generalization of convolution concept to graphs have been popularly applied to many learning tasks such as estimating quantum chemical computation, and modeling physical systems, etc. Spectral methods such as (Bruna et al., 2014) provide one way to define convolution on graphs via convolution theorem and graph Fourier transform (GFT). To address the high computational cost of GFT, (Xu et al., 2019) proposed to use the diffusion wavelet bases as previously defined by (Coifman and Maggioni, 2006) instead for a faster transformation.

6.3 Notation

We define $[n] = \{1, 2, \dots, n\}$ as the set of the first n natural numbers. We denote \mathbf{I}_n as the n dimensional identity matrix. The group of n dimensional orthogonal matrices is $\mathbb{SO}(n)$. $\mathbb{A} \cup \mathbb{B}$ will denote the disjoint union of two sets \mathbb{A} and \mathbb{B} , therefore $\mathbb{A}_1 \cup \mathbb{A}_2 \cup \dots \cup \mathbb{A}_k = \mathbb{S}$ is a

partition of \mathbb{S} .

Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and two sequences of indices $\mathbf{i} = (i_1, \dots, i_k) \in [n]^k$ and $\mathbf{j} = (j_1, \dots, j_k) \in [n]^k$ assuming that $i_1 < i_2 < \dots < i_k$ and $j_1 < j_2 < \dots < j_k$, $\mathbf{A}_{\mathbf{i}, \mathbf{j}}$ will be the $k \times k$ matrix with entries $[\mathbf{A}_{\mathbf{i}, \mathbf{j}}]_{x,y} = \mathbf{A}_{i_x, j_y}$. Furthermore, $\mathbf{A}_{i,:}$ and $\mathbf{A}_{:,j}$ denote the i -th row and the j -th column of \mathbf{A} , respectively. Given $\mathbf{A}_1 \in \mathbb{R}^{n_1 \times m_1}$ and $\mathbf{A}_2 \in \mathbb{R}^{n_2 \times m_2}$, $\mathbf{A}_1 \oplus \mathbf{A}_2$ is the $(n_1 + n_2) \times (m_1 + m_2)$ dimensional matrix with entries

$$[\mathbf{A}_1 \oplus \mathbf{A}_2]_{i,j} = \begin{cases} [\mathbf{A}_1]_{i,j} & \text{if } i \leq n_1 \text{ and } j \leq m_1 \\ [\mathbf{A}_2]_{i-n_1, j-m_1} & \text{if } i > n_1 \text{ and } j > m_1 \\ 0 & \text{otherwise.} \end{cases}$$

A matrix \mathbf{A} is said to be block diagonal if it is of the form

$$\mathbf{A} = \mathbf{A}_1 \oplus \mathbf{A}_2 \oplus \cdots \oplus \mathbf{A}_p \quad (6.1)$$

for some sequence of smaller matrices $\mathbf{A}_1, \dots, \mathbf{A}_p$. For the generalized block diagonal matrix, we remove the restriction that each block in (6.1) must involve a contiguous set of indices, and introduce the notation

$$\mathbf{A} = \oplus_{(i_1^1, \dots, i_{k_1}^1)} \mathbf{A}_1 \oplus_{(i_1^2, \dots, i_{k_2}^2)} \mathbf{A}_2 \cdots \oplus_{(i_1^p, \dots, i_{k_p}^p)} \mathbf{A}_p$$

in which

$$\mathbf{A}_{a,b} = \begin{cases} [\mathbf{A}_u]_{q,r} & \text{if } i_q^u = a \text{ and } i_r^u = b \text{ for some } u, q, r, \\ 0 & \text{otherwise.} \end{cases}$$

The Kronecker tensor product $\mathbf{A}_1 \otimes \mathbf{A}_2$ is an $n_1 n_2 \times m_1 m_2$ matrix whose elements are

$$[\mathbf{A}_1 \otimes \mathbf{A}_2]_{(i_1-1)n_2+i_2, (j_1-1)m_2+j_2} = [\mathbf{A}_1]_{i_1, j_1} \cdot [\mathbf{A}_2]_{i_2, j_2},$$

with the obvious generalization to p -fold products $\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_p$. We denote $\mathbf{A}^{\otimes p}$ as the p -fold product $\mathbf{A} \otimes \mathbf{A} \otimes \cdots \otimes \mathbf{A}$.

A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called skew-symmetric (or anti-symmetric) if $\mathbf{A}^T = -\mathbf{A}$. The Euclidean inner product between two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{m \times n}$ is defined as

$$\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{j,k} \mathbf{A}_{j,k} \mathbf{B}_{j,k} = \text{trace}(\mathbf{A}^T \mathbf{B}).$$

The Frobenius norm of \mathbf{A} is defined as $\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} \mathbf{A}_{i,j}^2}$.

6.4 Multiresolution Matrix Factorization

6.4.1 Background

Most commonly used matrix factorization algorithms, such as principal component analysis (PCA), singular value decomposition (SVD), or non-negative matrix factorization (NMF) are inherently single-level algorithms. Saying that a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is of rank $r \ll n$ means that it can be expressed in terms of a dictionary of r mutually orthogonal unit vectors $\{u_1, u_2, \dots, u_r\}$ in the form

$$\mathbf{A} = \sum_{i=1}^r \lambda_i u_i u_i^T,$$

where u_1, \dots, u_r are the normalized eigenvectors of \mathbf{A} and $\lambda_1, \dots, \lambda_r$ are the corresponding eigenvalues. This is the decomposition that PCA finds, and it corresponds to factorizing \mathbf{A}

in the form

$$\mathbf{A} = \mathbf{U}^T \mathbf{H} \mathbf{U}, \quad (6.2)$$

where \mathbf{U} is an orthogonal matrix and \mathbf{H} is a diagonal matrix with the eigenvalues of \mathbf{A} on its diagonal. The drawback of PCA is that eigenvectors are almost always dense, while matrices occurring in learning problems, especially those related to graphs, often have strong locality properties, in the sense that they are more closely couple certain clusters of nearby coordinates than those farther apart with respect to the underlying topology. In such cases, modeling A in terms of a basis of global eigenfunctions is both computationally wasteful and conceptually unreasonable: a localized dictionary would be more appropriate. In contrast to PCA, (Kondor et al., 2014) proposed *Multiresolution Matrix Factorization*, or MMF for short, to construct a sparse hierarchical system of L -level dictionaries. The corresponding matrix factorization is of the form

$$\mathbf{A} = \mathbf{U}_1^T \mathbf{U}_2^T \dots \mathbf{U}_L^T \mathbf{H} \mathbf{U}_L \dots \mathbf{U}_2 \mathbf{U}_1,$$

where \mathbf{H} is close to diagonal and $\mathbf{U}_1, \dots, \mathbf{U}_L$ are sparse orthogonal matrices with the following constraints:

1. Each \mathbf{U}_ℓ is k -point rotation for some small k , meaning that it only rotates k coordinates at a time. Formally, Def. 6.4.1 defines and Fig. 6.2 shows an example of the k -point rotation matrix.
2. There is a nested sequence of sets $\mathbb{S}_L \subseteq \dots \subseteq \mathbb{S}_1 \subseteq \mathbb{S}_0 = [n]$ such that the coordinates rotated by \mathbf{U}_ℓ are a subset of \mathbb{S}_ℓ .
3. \mathbf{H} is an \mathbb{S}_L -core-diagonal matrix that is formally defined in Def. 6.4.2.

Definition 6.4.1. We say that $\mathbf{U} \in \mathbb{R}^{n \times n}$ is an **elementary rotation of order k** (also

called as a k -point rotation) if it is an orthogonal matrix of the form

$$\mathbf{U} = \mathbf{I}_{n-k} \oplus_{(i_1, \dots, i_k)} \mathbf{O}$$

for some $\mathbb{I} = \{i_1, \dots, i_k\} \subseteq [n]$ and $\mathbf{O} \in \mathbb{SO}(k)$. We denote the set of all such matrices as $\mathbb{SO}_k(n)$.

The simplest case are second order rotations, or called Givens rotations, which are of the form

$$\mathbf{U} = \mathbf{I}_{n-2} \oplus_{(i,j)} \mathbf{O} = \begin{pmatrix} \cdot & & & \\ & \cos(\theta) & -\sin(\theta) & \\ & \cdot & \cdot & \\ & \sin(\theta) & \cos(\theta) & \\ & & & \cdot \end{pmatrix}, \quad (6.3)$$

where the dots denote the identity that apart from rows/columns i and j , and $\mathbf{O} \in \mathbb{SO}(2)$ is the rotation matrix of some angle $\theta \in [0, 2\pi)$. Indeed, Jacobi's algorithm for diagonalizing symmetric matrices (Jacobi, 1846) is a special case of MMF factorization over Givens rotations.

Definition 6.4.2. Given a set $\mathbb{S} \subseteq [n]$, we say that a matrix $\mathbf{H} \in \mathbb{R}^{n \times n}$ is \mathbb{S} -core-diagonal if $\mathbf{H}_{i,j} = 0$ unless $i, j \in \mathbb{S}$ or $i = j$. Equivalently, \mathbf{H} is \mathbb{S} -core-diagonal if it can be written in the form $\mathbf{H} = \mathbf{D} \oplus_{\mathbb{S}} \overline{\mathbf{H}}$, for some $\overline{\mathbf{H}} \in \mathbb{R}^{|\mathbb{S}| \times |\mathbb{S}|}$ and \mathbf{D} is diagonal. We denote the set of all \mathbb{S} -core-diagonal symmetric matrices of dimension n as $\mathbb{H}_n^{\mathbb{S}}$.

6.4.2 Multiresolution analysis

We formally define MMF in Defs. 6.4.3 and 6.4.4. Furthermore, (Kondor et al., 2014) has shown that MMF mirrors the classical theory of multiresolution analysis (MRA) on the real line (Mallat, 1989b) to discrete spaces. The functional analytic view of wavelets is provided by MRA, which, similarly to Fourier analysis, is a way of filtering some function space into a sequence of subspaces

$$\dots \subset \mathbb{V}_{-1} \subset \mathbb{V}_0 \subset \mathbb{V}_1 \subset \mathbb{V}_2 \subset \dots \quad (6.4)$$

Definition 6.4.3. Given an appropriate subset \mathbb{O} of the group $\mathbb{SO}(n)$ of n -dimensional rotation matrices, a depth parameter $L \in \mathbb{N}$, and a sequence of integers $n = d_0 \geq d_1 \geq d_2 \geq \dots \geq d_L \geq 1$, a **Multiresolution Matrix Factorization (MMF)** of a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ over \mathbb{O} is a factorization of the form

$$\mathbf{A} = \mathbf{U}_1^T \mathbf{U}_2^T \dots \mathbf{U}_L^T \mathbf{H} \mathbf{U}_L \dots \mathbf{U}_2 \mathbf{U}_1, \quad (6.5)$$

where each $\mathbf{U}_\ell \in \mathbb{O}$ satisfies $[\mathbf{U}_\ell]_{[n] \setminus \mathbb{S}_{\ell-1}, [n] \setminus \mathbb{S}_{\ell-1}} = \mathbf{I}_{n-d_\ell}$ for some nested sequence of sets $\mathbb{S}_L \subseteq \dots \subseteq \mathbb{S}_1 \subseteq \mathbb{S}_0 = [n]$ with $|\mathbb{S}_\ell| = d_\ell$, and $\mathbf{H} \in \mathbb{H}_n^{\mathbb{S}_L}$ is an \mathbb{S}_L -core-diagonal matrix.

Definition 6.4.4. We say that a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is **fully multiresolution factorizable** over $\mathbb{O} \subset \mathbb{SO}(n)$ with (d_1, \dots, d_L) if it has a decomposition of the form described in Def. 6.4.3.

However, it is best to conceptualize (6.4) as an iterative process of splitting each \mathbb{V}_ℓ into the orthogonal sum $\mathbb{V}_\ell = \mathbb{V}_{\ell+1} \oplus \mathbb{W}_{\ell+1}$ of a smoother part $\mathbb{V}_{\ell+1}$, called the *approximation space*; and a rougher part $\mathbb{W}_{\ell+1}$, called the *detail space* (see Fig. 6.1). Each \mathbb{V}_ℓ has an orthonormal basis $\Phi_\ell \triangleq \{\phi_m^\ell\}_m$ in which each ϕ is called a *father* wavelet. Each complementary space \mathbb{W}_ℓ is also spanned by an orthonormal basis $\Psi_\ell \triangleq \{\psi_m^\ell\}_m$ in which each ψ is called a *mother* wavelet. In MMF, each individual rotation $\mathbf{U}_\ell : \mathbb{V}_{\ell-1} \rightarrow \mathbb{V}_\ell \oplus \mathbb{W}_\ell$ is a sparse basis transform

that expresses $\Phi_\ell \cup \Psi_\ell$ in the previous basis $\Phi_{\ell-1}$ such that:

$$\phi_m^\ell = \sum_{i=1}^{\dim(\mathbb{V}_{\ell-1})} [\mathbf{U}_\ell]_{m,i} \phi_i^{\ell-1},$$

$$\psi_m^\ell = \sum_{i=1}^{\dim(\mathbb{V}_{\ell-1})} [\mathbf{U}_\ell]_{m+\dim(\mathbb{V}_{\ell-1}),i} \phi_i^{\ell-1},$$

in which Φ_0 is the standard basis, i.e. $\phi_m^0 = e_m$; and $\dim(\mathbb{V}_\ell) = d_\ell = |\mathbb{S}_\ell|$. In the $\Phi_1 \cup \Psi_1$ basis, \mathbf{A} compresses into $\mathbf{A}_1 = \mathbf{U}_1 \mathbf{A} \mathbf{U}_1^T$. In the $\Phi_2 \cup \Psi_2 \cup \Psi_1$ basis, it becomes $\mathbf{A}_2 = \mathbf{U}_2 \mathbf{U}_1 \mathbf{A} \mathbf{U}_1^T \mathbf{U}_2^T$, and so on. Finally, in the $\Phi_L \cup \Psi_L \cup \dots \cup \Psi_1$ basis, it takes on the form $\mathbf{A}_L = \mathbf{H} = \mathbf{U}_L \dots \mathbf{U}_2 \mathbf{U}_1 \mathbf{A} \mathbf{U}_1^T \mathbf{U}_2^T \dots \mathbf{U}_L^T$ that consists of four distinct blocks (supposingly that we permute the rows/columns accordingly):

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_{\Phi,\Phi} & \mathbf{H}_{\Phi,\Psi} \\ \mathbf{H}_{\Psi,\Phi} & \mathbf{H}_{\Psi,\Psi} \end{pmatrix},$$

where $\mathbf{H}_{\Phi,\Phi} \in \mathbb{R}^{\dim(\mathbb{V}_L) \times \dim(\mathbb{V}_L)}$ is effectively \mathbf{A} compressed to \mathbb{V}_L , $\mathbf{H}_{\Phi,\Psi} = \mathbf{H}_{\Psi,\Phi}^T = 0$ and $\mathbf{H}_{\Psi,\Psi}$ is diagonal. MMF approximates \mathbf{A} in the form

$$\mathbf{A} \approx \sum_{i,j=1}^{d_L} h_{i,j} \phi_i^L \phi_j^{L^T} + \sum_{\ell=1}^L \sum_{m=1}^{d_\ell} c_m^\ell \psi_m^\ell \psi_m^{\ell T},$$

where $h_{i,j}$ coefficients are the entries of the $\mathbf{H}_{\Phi,\Phi}$ block, and $c_m^\ell = \langle \psi_m^\ell, \mathbf{A} \psi_m^\ell \rangle$ wavelet frequencies are the diagonal elements of the $\mathbf{H}_{\Psi,\Psi}$ block.

In particular, the dictionary vectors corresponding to certain rows of \mathbf{U}_1 are interpreted as level one wavelets, the dictionary vectors corresponding to certain rows of $\mathbf{U}_2 \mathbf{U}_1$ are interpreted as level two wavelets, and so on (see Section 6.4.2). One thing that is immediately clear is that whereas Eq. (6.2) diagonalizes \mathbf{A} in a single step, multiresolution analysis will

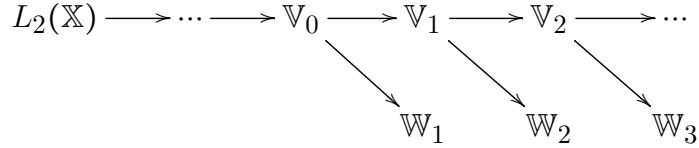


Figure 6.1: Multiresolution analysis splits each function space $\mathbb{V}_0, \mathbb{V}_1, \dots$ into the direct sum of a smoother part $\mathbb{V}_{\ell+1}$ and a rougher part $\mathbb{W}_{\ell+1}$.

$$\mathbf{I}_{n-k} \oplus_{(i_1, \dots, i_k)} \mathbf{O} = \Pi \begin{pmatrix} & & \\ & \blacksquare & \\ & & \end{pmatrix} \Pi^\top$$

Figure 6.2: A rotation matrix of order k . The purpose of permutation matrix Π is solely to ensure that the blocks of the matrices appear contiguous in the figure. In this case, $n = 17$ and $k = 4$.

involve a sequence of basis transforms $\mathbf{U}_1, \mathbf{U}_2, \dots, \mathbf{U}_L$, transforming \mathbf{A} step by step as

$$\mathbf{A} \rightarrow \mathbf{U}_1 \mathbf{A} \mathbf{U}_1^T \rightarrow \mathbf{U}_2 \mathbf{U}_1 \mathbf{A} \mathbf{U}_1^T \mathbf{U}_2^T \rightarrow \cdots \rightarrow \mathbf{U}_L \dots \mathbf{U}_2 \mathbf{U}_1 \mathbf{A} \mathbf{U}_1^T \mathbf{U}_2^T \dots \mathbf{U}_L^T, \quad (6.6)$$

so the corresponding matrix factorization must be a multilevel factorization

$$\mathbf{A} \approx \mathbf{U}_1^T \mathbf{U}_2^T \dots \mathbf{U}_\ell^T \mathbf{H} \mathbf{U}_\ell \dots \mathbf{U}_2 \mathbf{U}_1. \quad (6.7)$$

Fig. 6.3 depicts the multiresolution transform of MMF as in Eq. (6.6). Fig. 6.4 illustrates the corresponding factorization as in Eq. (6.7).

6.4.3 Optimization

Finding the best MMF factorization to a symmetric matrix \mathbf{A} involves solving

$$\min_{\substack{\mathbb{S}_L \subseteq \dots \subseteq \mathbb{S}_1 \subseteq \mathbb{S}_0 = [n] \\ \mathbf{H} \in \mathbb{H}_n^{\mathbb{S}_L}; \mathbf{U}_1, \dots, \mathbf{U}_L \in \mathcal{O}}} \|\mathbf{A} - \mathbf{U}_1^T \dots \mathbf{U}_L^T \mathbf{H} \mathbf{U}_L \dots \mathbf{U}_1\|. \quad (6.8)$$

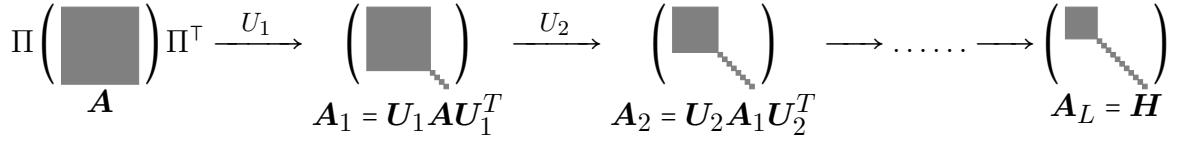


Figure 6.3: MMF can be thought of as a process of successively compressing \mathbf{A} to size $d_1 \times d_1$, $d_2 \times d_2$, etc. (plus the diagonal entries) down to the final $d_L \times d_L$ core-diagonal matrix \mathbf{H} (see Def. 6.4.3). The role of permutation matrix Π is purely for the ease of visualization (as in Fig. 6.2).

$$\Pi \begin{pmatrix} \text{large gray square} \\ A \end{pmatrix} \Pi^\top \approx \begin{pmatrix} \text{small gray square} \\ U_1^T \end{pmatrix} \dots \begin{pmatrix} \text{small gray square} \\ U_L^T \end{pmatrix} \begin{pmatrix} \text{small gray square} \\ H \end{pmatrix} \begin{pmatrix} \text{small gray square} \\ U_L \end{pmatrix} \dots \begin{pmatrix} \text{small gray square} \\ U_1 \end{pmatrix}$$

Figure 6.4: Matrix approximation as in Eq. 6.5. In this figure, the core block size of each rotation matrix \mathbf{U}_ℓ and \mathbf{H} are $k \times k = 4 \times 4$ and $d_L \times d_L = 8 \times 8$, respectively. Permutation matrix Π is only for visualization (as in Figs. 6.2 6.3).

Assuming that we measure error in the Frobenius norm, (6.8) is equivalent to

$$\min_{\substack{\mathbb{S}_L \subseteq \dots \subseteq \mathbb{S}_1 \subseteq \mathbb{S}_0 = [n] \\ \mathbf{U}_1, \dots, \mathbf{U}_L \in \mathbb{O}}} \|\mathbf{U}_L \dots \mathbf{U}_1 \mathbf{A} \mathbf{U}_1^T \dots \mathbf{U}_L^T\|_{\text{resi}}^2, \quad (6.9)$$

where $\|\cdot\|_{\text{resi}}^2$ is the squared residual norm $\|\mathbf{H}\|_{\text{resi}}^2 = \sum_{i \neq j; (i,j) \notin \mathbb{S}_L \times \mathbb{S}_L} |\mathbf{H}_{i,j}|^2$.

Heuristically, factorizing \mathbf{A} can be approximated by an iterative process that starts by setting $\mathbf{A}_0 = \mathbf{A}$ and $\mathbb{S}_1 = [n]$, and then executes the following steps for each resolution level $\ell \in \{1, \dots, L\}$:

1. Given $\mathbf{A}_{\ell-1}$, select k indices $\mathbb{I}_\ell = \{i_1, \dots, i_k\} \subset \mathbb{S}_{\ell-1}$ of rows/columns of the active submatrix $[\mathbf{A}_{\ell-1}]_{\mathbb{S}_{\ell-1}, \mathbb{S}_{\ell-1}}$ that are highly correlated with each other.
2. Find the corresponding k -point rotation \mathbf{U}_ℓ to \mathbb{I}_ℓ , and compute $\mathbf{A}_\ell = \mathbf{U}_\ell \mathbf{A}_{\ell-1} \mathbf{U}_\ell^T$ that brings the submatrix $[\mathbf{A}_{\ell-1}]_{\mathbb{I}_\ell, \mathbb{I}_\ell}$ close to diagonal. In the last level, we set $\mathbf{H} = \mathbf{A}_L$ (see Fig. 6.3).
3. Determine the set of coordinates $\mathbb{T}_\ell \subseteq \mathbb{S}_{\ell-1}$ that are to be designated wavelets at this

level, and eliminate them from the active set by setting $\mathbb{S}_\ell = \mathbb{S}_{\ell-1} \setminus \mathbb{T}_\ell$.

6.4.4 Proposal of a learning algorithm

There are two fundamental difficulties in MMF optimization: finding the optimal nested sequence of \mathbb{S}_ℓ is a combinatorially hard (e.g., there are $\binom{d_\ell}{k}$ ways to choose k indices out of \mathbb{S}_ℓ); and the solution for \mathbf{U}_ℓ must satisfy the orthogonality constraint such that $\mathbf{U}_\ell^T \mathbf{U}_\ell = \mathbf{I}$. The existing literature on solving this optimization problem (Kondor et al., 2014) (Teneva et al., 2016) (Ithapu et al., 2017) (Ding et al., 2017) has various heuristic elements and has a number of limitations:

- There is no guarantee that the greedy heuristics (e.g., clustering) used in selecting k rows/columns $\mathbb{I}_\ell = \{i_1, \dots, i_k\} \subset \mathbb{S}_\ell$ for each rotation return a globally optimal factorization.
- Instead of direct optimization for each rotation $\mathbf{U}_\ell \triangleq \mathbf{I}_{n-k} \oplus_{\mathbb{I}_\ell} \mathbf{O}_\ell$ where $\mathbf{O}_\ell \in \mathbb{SO}(k)$ globally and simultaneously with the objective (6.8), Jacobi MMFs (see Proposition 2 of (Kondor et al., 2014)) apply the greedy strategy of optimizing them locally and sequentially. Again, this does not necessarily lead to a *globally* optimal combination of rotations.
- Most MMF algorithms are limited to the simplest case of $k = 2$ where \mathbf{U}_ℓ is just a Given rotation, which can be parameterized by a single variable, the rotation angle θ_ℓ . This makes it possible to optimize the greed objective by simple gradient descent, but larger rotations would yield more expressive factorizations and better approximations.

In contrast, we propose an iterative algorithm to directly optimize the global MMF objective (6.8):

- We use gradient descent algorithm on the Stiefel manifold to optimize all rotations $\{\mathbf{U}_\ell\}_{\ell=1}^L$ *simultaneously*, whilst satisfying the orthogonality constraints. Importantly, the Stiefel manifold optimization is not limited to $k = 2$ case (Section 6.5).

- We formulate the problem of finding the optimal nested sequence $\mathbb{S}_L \subseteq \dots \subseteq \mathbb{S}_1 \subseteq \mathbb{S}_0 = [n]$ as learning a Markov Decision Process (MDP) that can be subsequently solved by the gradient policy method of Reinforcement Learning (RL), in which the RL agent (or stochastic policy) is modeled by graph neural networks (GNN) (Section 6.6).

We show that the resulting learning-based MMF algorithm outperforms existing greedy MMFs and other traditional baselines for matrix approximation in various scenarios (see Section 6.8).

6.5 Stiefel Manifold Optimization

The MMF optimization problem in (6.8) and (6.9) is equivalent to

$$\min_{\mathbb{S}_L \subseteq \dots \subseteq \mathbb{S}_1 \subseteq \mathbb{S}_0 = [n]} \min_{\mathbf{U}_1, \dots, \mathbf{U}_L \in \mathbb{O}} \|\mathbf{U}_L \dots \mathbf{U}_1 \mathbf{A} \mathbf{U}_1^T \dots \mathbf{U}_L^T\|_{\text{resi}}^2, \quad (6.10)$$

In order to solve the inner optimization problem of (6.10), we consider the following generic optimization with orthogonality constraints:

$$\min_{\mathbf{X} \in \mathbb{R}^{n \times p}} \mathcal{F}(\mathbf{X}), \quad \text{s.t. } \mathbf{X}^T \mathbf{X} = \mathbf{I}_p, \quad (6.11)$$

where \mathbf{I}_p is the identity matrix and $\mathcal{F}(\mathbf{X}) : \mathbb{R}^{n \times p} \rightarrow \mathbb{R}$ is a differentiable function. The feasible set $\mathcal{V}_p(\mathbb{R}^n) = \{\mathbf{X} \in \mathbb{R}^{n \times p} : \mathbf{X}^T \mathbf{X} = \mathbf{I}_p\}$ is referred to as the Stiefel manifold of p orthonormal vectors in \mathbb{R}^n that has dimension equal to $np - \frac{1}{2}p(p+1)$. We will view $\mathcal{V}_p(\mathbb{R}^n)$ as an embedded submanifold of $\mathbb{R}^{n \times p}$.

When there is more than one orthogonal constraint, (6.11) is written as

$$\min_{\mathbf{X}_1 \in \mathcal{V}_{p_1}(\mathbb{R}^{n_1}), \dots, \mathbf{X}_q \in \mathcal{V}_{p_q}(\mathbb{R}^{n_q})} \mathcal{F}(\mathbf{X}_1, \dots, \mathbf{X}_q) \quad (6.12)$$

where there are q variables with corresponding q orthogonal constraints. For example, in the MMF optimization problem (6.8), suppose we are already given $\mathbb{S}_L \subseteq \dots \subseteq \mathbb{S}_1 \subseteq \mathbb{S}_0 = [n]$ meaning that the indices of active rows/columns at each resolution were already determined, for simplicity. In this case, we have $q = L$ number of variables such that each variable $\mathbf{X}_\ell = \mathbf{O}_\ell \in \mathbb{R}^{k \times k}$, where $\mathbf{U}_\ell = \mathbf{I}_{n-k} \oplus \mathbb{I}_\ell \mathbf{O}_\ell \in \mathbb{R}^{n \times n}$ in which \mathbb{I}_ℓ is a subset of k indices from \mathbb{S}_ℓ , must satisfy the orthogonality constraint. The corresponding objective function is

$$\mathcal{F}(\mathbf{O}_1, \dots, \mathbf{O}_L) = \|\mathbf{U}_L \dots \mathbf{U}_1 \mathbf{A} \mathbf{U}_1^T \dots \mathbf{U}_L^T\|_{\text{resi}}^2. \quad (6.13)$$

We identify tangent vectors to the manifold with $n \times p$ matrices. We denote the tangent space at \mathbf{X} as $\mathcal{T}_{\mathbf{X}} \mathcal{V}_p(\mathbb{R}^n)$. Lemma 6.5.1 characterizes vectors in the tangent space.

Lemma 6.5.1. *Any $\mathbf{Z} \in \mathcal{T}_{\mathbf{X}} \mathcal{V}_p(\mathbb{R}^n)$, then \mathbf{Z} (as an element of $\mathbb{R}^{n \times p}$) satisfies*

$$\mathbf{Z}^T \mathbf{X} + \mathbf{X}^T \mathbf{Z} = 0,$$

where $\mathbf{Z}^T \mathbf{X}$ is a skew-symmetric $p \times p$ matrix.

Proof. Let $\mathbf{Y}(t)$ be a curve in $\mathcal{V}_p(\mathbb{R}^n)$ that starts from \mathbf{X} . We have:

$$\mathbf{Y}^T(t) \mathbf{Y}(t) = \mathbf{I}_p. \quad (6.14)$$

We differentiate two sides of Eq. (6.14) with respect to t :

$$\frac{d}{dt}(\mathbf{Y}^T(t) \mathbf{Y}(t)) = 0$$

that leads to:

$$\left(\frac{d\mathbf{Y}}{dt}(0) \right)^T \mathbf{Y}(0) + \mathbf{Y}(0)^T \frac{d\mathbf{Y}}{dt}(0) = 0$$

at $t = 0$. Recall that by definition, $\mathbf{Y}(0) = \mathbf{X}$ and $\frac{d\mathbf{Y}}{dt}(0)$ is any element of the tangent space

at \mathbf{X} . Therefore, we arrive at $\mathbf{Z}^T \mathbf{X} + \mathbf{X}^T \mathbf{Z} = 0$. \square

Suppose that \mathcal{F} is a differentiable function. The gradient of \mathcal{F} with respect to \mathbf{X} is denoted by $\mathbf{G} \triangleq \mathcal{D}\mathcal{F}_{\mathbf{X}} \triangleq \left(\frac{\partial \mathcal{F}(\mathbf{X})}{\partial \mathbf{X}_{i,j}} \right)$. The derivative of \mathcal{F} at \mathbf{X} in a direction \mathbf{Z} is

$$\mathcal{D}\mathcal{F}_{\mathbf{X}}(\mathbf{Z}) \triangleq \lim_{t \rightarrow 0} \frac{\mathcal{F}(\mathbf{X} + t\mathbf{Z}) - \mathcal{F}(\mathbf{X})}{t} = \langle \mathbf{G}, \mathbf{Z} \rangle$$

Since the matrix $\mathbf{X}^T \mathbf{X}$ is symmetric, the Lagrangian multiplier Λ corresponding to $\mathbf{X}^T \mathbf{X} = \mathbf{I}_p$ is a symmetric matrix. The Lagrangian function of problem (6.11) is

$$\mathcal{L}(\mathbf{X}, \Lambda) = \mathcal{F}(\mathbf{X}) - \frac{1}{2} \text{trace}(\Lambda(\mathbf{X}^T \mathbf{X} - \mathbf{I}_p)) \quad (6.15)$$

Lemma 6.5.2. Suppose that \mathbf{X} is a local minimizer of problem (6.11). Then \mathbf{X} satisfies the first-order optimality conditions $\mathcal{D}_{\mathbf{X}} \mathcal{L}(\mathbf{X}, \Lambda) = \mathbf{G} - \mathbf{X} \mathbf{G}^T \mathbf{X} = 0$ and $\mathbf{X}^T \mathbf{X} = \mathbf{I}_p$ with the associated Lagrangian multiplier $\Lambda = \mathbf{G}^T \mathbf{X}$. Define $\nabla \mathcal{F}(\mathbf{X}) \triangleq \mathbf{G} - \mathbf{X} \mathbf{G}^T \mathbf{X}$ and $\mathbf{A} \triangleq \mathbf{G} \mathbf{X}^T - \mathbf{X} \mathbf{G}^T$. Then $\nabla \mathcal{F} = \mathbf{A} \mathbf{X}$. Moreover, $\nabla \mathcal{F} = 0$ if and only if $\mathbf{A} = 0$.

Proof. Since $\mathbf{X} \in \mathcal{V}_p(\mathbb{R}^n)$, we have $\mathbf{X}^T \mathbf{X} = \mathbf{I}_p$. We differentiate both sides of the Lagrangian function:

$$\mathcal{D}_{\mathbf{X}} \mathcal{L}(\mathbf{X}, \Lambda) = \mathcal{D}\mathcal{F}(\mathbf{X}) - \mathbf{X} \Lambda = 0.$$

Recall that by definition, $\mathbf{G} \triangleq \mathcal{D}\mathcal{F}(\mathbf{X})$, we have

$$\mathcal{D}_{\mathbf{X}} \mathcal{L}(\mathbf{X}, \Lambda) = \mathbf{G} - \mathbf{X} \Lambda = 0. \quad (6.16)$$

Multiplying both sides by \mathbf{X}^T , we get $\mathbf{X}^T \mathbf{G} - \mathbf{X}^T \mathbf{X} \Lambda = 0$ that leads to $\mathbf{X}^T \mathbf{G} - \Lambda = 0$ or $\Lambda = \mathbf{X}^T \mathbf{G}$. Since the matrix $\mathbf{X}^T \mathbf{X}$ is symmetric, the Lagrangian multiplier Λ corresponding to $\mathbf{X}^T \mathbf{X} = \mathbf{I}_p$ is a symmetric matrix. Therefore, we obtain $\Lambda = \Lambda^T = \mathbf{G}^T \mathbf{X}$ and $\mathcal{D}_{\mathbf{X}} \mathcal{L}(\mathbf{X}, \Lambda) = \mathbf{G} - \mathbf{X} \mathbf{G}^T \mathbf{X} = 0$. By definition, $\mathbf{A} \triangleq \mathbf{G} \mathbf{X}^T - \mathbf{X} \mathbf{G}^T$. We have $\mathbf{A} \mathbf{X} = \mathbf{G} - \mathbf{X} \mathbf{G}^T \mathbf{X} = \nabla \mathcal{F}$. The

last statement is trivial. \square

Let $\mathbf{X} \in \mathcal{V}_p(\mathbb{R}^n)$, and \mathbf{W} be any $n \times n$ skew-symmetric matrix. We consider the following curve that transforms \mathbf{X} by $(\mathbf{I} + \frac{\tau}{2}\mathbf{W})^{-1}(\mathbf{I} - \frac{\tau}{2}\mathbf{W})$:

$$\mathbf{Y}(\tau) = (\mathbf{I} + \frac{\tau}{2}\mathbf{W})^{-1}(\mathbf{I} - \frac{\tau}{2}\mathbf{W})\mathbf{X}. \quad (6.17)$$

This is called as the *Cayley transformation*. Its derivative with respect to τ is

$$\mathbf{Y}'(\tau) = -\left(\mathbf{I} + \frac{\tau}{2}\mathbf{W}\right)^{-1}\mathbf{W}\left(\frac{\mathbf{X} + \mathbf{Y}(\tau)}{2}\right). \quad (6.18)$$

The curve has the following properties:

1. It stays in the Stiefel manifold, i.e. $\mathbf{Y}(\tau)^T\mathbf{Y}(\tau) = \mathbf{I}$.
2. Its tangent vector at $\tau = 0$ is $\mathbf{Y}'(0) = -\mathbf{W}\mathbf{X}$. It can be easily derived from Lemma 6.5.1 that $\mathbf{Y}'(0)$ is in the tangent space $\mathcal{T}_{\mathbf{Y}(0)}\mathcal{V}_p(\mathbb{R}^n)$. Since $\mathbf{Y}(0) = \mathbf{X}$ and \mathbf{W} is a skew-symmetric matrix, by letting $\mathbf{Z} = -\mathbf{W}\mathbf{X}$, it is trivial that $\mathbf{Z}^T\mathbf{X} + \mathbf{X}^T\mathbf{Z} = 0$.

Lemma 6.5.3. *If we set $\mathbf{W} \triangleq \mathbf{A} \triangleq \mathbf{G}\mathbf{X}^T - \mathbf{X}\mathbf{G}^T$ (see Lemma 6.5.2), then the curve $\mathbf{Y}(\tau)$ (defined in Eq. (6.17)) is a decent curve for \mathcal{F} at $\tau = 0$, that is*

$$\mathcal{F}'_\tau(\mathbf{Y}(0)) \triangleq \frac{\partial \mathcal{F}(\mathbf{Y}(\tau))}{\partial \tau} \Big|_{\tau=0} = -\frac{1}{2}\|\mathbf{A}\|_F^2.$$

Proof. By the chain rule, we get

$$\mathcal{F}'_\tau(\mathbf{Y}(\tau)) = \text{trace}(\mathcal{D}\mathcal{F}(\mathbf{Y}(\tau))^T\mathbf{Y}'(\tau)).$$

At $\tau = 0$, $\mathcal{D}\mathcal{F}(\mathbf{Y}(0)) = \mathbf{G}$ and $\mathbf{Y}'(0) = -\mathbf{A}\mathbf{X}$. Therefore,

$$\mathcal{F}'_\tau(\mathbf{Y}(0)) = -\text{trace}(\mathbf{G}^T(\mathbf{G}\mathbf{X}^T - \mathbf{X}\mathbf{G}^T)\mathbf{X}) = -\frac{1}{2}\text{trace}(\mathbf{A}\mathbf{A}^T) = -\frac{1}{2}\|\mathbf{A}\|_F^2.$$

□

It is well known that the steepest descent method with a fixed step size may not converge, but the convergence can be guaranteed by choosing the step size wisely: one can choose a step size by minimizing $\mathcal{F}(\mathbf{Y}(\tau))$ along the curve $\mathbf{Y}(\tau)$ with respect to τ (Wen and Yin, 2010). With the choice of \mathbf{W} given by Lemma 6.5.3, the minimization algorithm using $\mathbf{Y}(\tau)$ is roughly sketched as follows: Start with some initial $\mathbf{X}^{(0)}$. For $t > 0$, we generate $\mathbf{X}^{(t+1)}$ from $\mathbf{X}^{(t)}$ by a curvilinear search along the curve $\mathbf{Y}(\tau) = (\mathbf{I} + \frac{\tau}{2}\mathbf{W})^{-1}(\mathbf{I} - \frac{\tau}{2}\mathbf{W})\mathbf{X}^{(t)}$ by changing τ . Because finding the global minimizer is computationally infeasible, the search terminates when then Armijo-Wolfe conditions that indicate an approximate minimizer are satisfied. The Armijo-Wolfe conditions require two parameters $0 < \rho_1 < \rho_2 < 1$ (Nocedal and Wright, 2006) (Wen and Yin, 2010) (Tagare, 2011):

$$\mathcal{F}(\mathbf{Y}(\tau)) \leq \mathcal{F}(\mathbf{Y}(0)) + \rho_1\tau\mathcal{F}'_\tau(\mathbf{Y}(0)) \quad (6.19)$$

$$\mathcal{F}'_\tau(\mathbf{Y}(\tau)) \geq \rho_2\mathcal{F}'_\tau(\mathbf{Y}(0)) \quad (6.20)$$

where $\mathcal{F}'_\tau(\mathbf{Y}(\tau)) = \text{trace}(\mathbf{G}^T\mathbf{Y}'(\tau))$ while $\mathbf{Y}'(\tau)$ is computed as Eq. (6.18) and $\mathbf{Y}'(0) = -\mathbf{A}\mathbf{X}$. The gradient descent algorithm on Stiefel manifold to optimize the generic orthogonal-constraint problem (6.11) with the curvilinear search submodule is described in Algorithm 16, which is used as a submodule in part of our learning algorithm to solve the MMF in (6.8). The algorithm can be trivially extended to solve problems with multiple variables and constraints.

Algorithm 16 Stiefel manifold gradient descent algorithm

```

1: Given  $0 < \rho_1 < \rho_2 < 1$  and  $\epsilon > 0$ .
2: Given an initial point  $\mathbf{X}^{(0)} \in \mathcal{V}_p(\mathbb{R}^n)$ .
3:  $t \leftarrow 0$ 
4: while true do
5:    $\mathbf{G} \leftarrow \left( \frac{\partial \mathcal{F}(\mathbf{X}^{(t)})}{\partial \mathbf{X}_{i,j}^{(t)}} \right)$             $\triangleright$  Compute the gradient of  $\mathcal{F}$  w.r.t  $\mathbf{X}$  element-wise
6:    $\mathbf{A} \leftarrow \mathbf{G}\mathbf{X}^{(t)T} - \mathbf{X}^{(t)}\mathbf{G}^T$             $\triangleright$  See Lemma 2, 3
7:   Initialize  $\tau$  to a non-zero value.            $\triangleright$  Curvilinear search for the optimal step size
8:   while (6.19) and (6.20) are not satisfied do            $\triangleright$  Armijo-Wolfe conditions
9:      $\tau \leftarrow \frac{\tau}{2}$             $\triangleright$  Reduce the step size by half
10:  end while
11:   $\mathbf{X}^{(t+1)} \leftarrow \mathbf{Y}(\tau)$             $\triangleright$  Update by the Cayley transformation
12:  if  $\|\nabla \mathcal{F}(\mathbf{X}^{(t+1)})\| \leq \epsilon$  then            $\triangleright$  Stopping check. See Lemma 2.
13:    STOP
14:  else
15:     $t \leftarrow t + 1$ 
16:  end if
17: end while

```

6.6 Reinforcement Learning

6.6.1 Problem formulation

We formulate the problem of finding the optimal nested sequence of sets $\mathbb{S}_L \subseteq \dots \subseteq \mathbb{S}_1 \subseteq \mathbb{S}_0 = [n]$ as learning an RL agent in order to solve the MMF optimization in (6.8). There are two fundamental parts to index selection for each resolution level $\ell \in \{1, \dots, L\}$:

- Select k indices $\mathbb{I}_\ell = \{i_1, \dots, i_k\} \subset \mathbb{S}_{\ell-1}$ to construct the corresponding rotation matrix \mathbf{U}_ℓ (see Section 6.5).
- Select the set of indices $\mathbb{T}_\ell \subset \mathbb{S}_{\ell-1}$ of rows/columns that are to be wavelets at this level, and then be eliminated by setting $\mathbb{S}_\ell = \mathbb{S}_{\ell-1} \setminus \mathbb{T}_\ell$. To reduce the computational cost, we assume that each resolution level has only one row/column to be selected as the wavelet (e.g., a single wavelet) such that $|\mathbb{T}_\ell| = 1$. That means the cardinality of \mathbb{S}_ℓ reduces by 1 after each level, $d_\ell = n - \ell$, and size of the core block of \mathbf{H} is $(n-L) \times (n-L)$ that corresponds to exactly $n - L$ active rows/columns at the end.

6.6.2 Markov Decision Process

A key task for building our model is to specify our index selection procedure. We design an iterative index selection process and formulate it as a general decision process $M = (S, A, P, R, \gamma)$ as follows.

S is the set of states (or state space) that consists of all possible intermediate and final states in which each state $s \in S$ is a tuple of $(\bar{\mathbf{A}}, \mathbb{S}, \ell)$ where ℓ indicates the resolution level, \mathbb{S} indicates the set of active row/column indices, and $\bar{\mathbf{A}} = \mathbf{A}_{\mathbb{S}, \mathbb{S}}$ indicates the sub-matrix of \mathbf{A} with indices of rows and columns are from \mathbb{S} (e.g., $|\mathbb{S}| = d_\ell$, $\bar{\mathbf{A}} \in \mathbb{R}^{d_\ell \times d_\ell}$). We start at state $s_0 = (\mathbf{A}, [n], 0)$ where \mathbf{A} is the input matrix that MMF tries to factorize (e.g., no rows/columns removed yet) and $[n]$ indicates all rows/columns are still active. The set of terminal (final) states $S^* \subset S$ includes every state s^* that has $\ell = L$.

A is the set of actions that describe the modification made to current state at each time step. An action $a \in A$ validly applied to a non-terminal state $s = (\mathbf{A}_{\mathbb{S}, \mathbb{S}}, \mathbb{S}, \ell)$ is a tuple (\mathbb{I}, \mathbb{T}) where $\mathbb{I} = \{i_1, \dots, i_k\} \subset \mathbb{S}$ is the set of k indices corresponding to the rotation matrix $\mathbf{U}_{\ell+1}$, and $\mathbb{T} \subset \mathbb{S}$ is the set of wavelet indices to spit out at this level. This action transforms the state into the next one $s' = (\mathbf{A}_{\mathbb{S}', \mathbb{S}'}, \mathbb{S}', \ell+1)$ where $\mathbb{S}' = \mathbb{S} \setminus \mathbb{T}$ meaning the set of active indices gets shrunked further. The action is called invalid for the current state if and only if $\mathbb{I} \notin \mathbb{S}$ or $\mathbb{T} \notin \mathbb{S}$.

P is the transition dynamics that specifies the possible outcomes of carrying out an action at time t , $p(s_{\ell+1}|s_\ell, \dots, s_0, a_\ell)$, as a conditional probability on the sequence of previous states (s_0, \dots, s_ℓ) and the action a_ℓ applied to state s_ℓ . Basically, the RL environment carries out actions that obey the given action rules. Invalid actions proposed by the policy network are rejected and the state remains unchanged. The state transition distribution is

constructed as

$$p(s_{\ell+1}|s_\ell, \dots, s_0) = \sum_{a_\ell} p(a_\ell|s_\ell, \dots, s_0) p(s_{\ell+1}|s_\ell, \dots, s_0, a_\ell),$$

where $p(a_\ell|s_\ell, \dots, s_0)$ is represented as a parameterized policy network π_θ with learnable parameters θ . Markov Decision Process (MDP) requires the state transition dynamics to satisfy the Markov property: $p(s_{\ell+1}|s_\ell, \dots, s_0) = p(s_{\ell+1}|s_\ell)$. Under this property, the policy network $\pi_\theta(a_\ell|s_\ell)$ only needs the intermediate state s_ℓ to derive an action. The whole trajectory is always started by the same s_0 and finished by a terminal state after exactly L transitions as depicted as follows:

$$s_0 \xrightarrow{a \sim \pi_\theta(\cdot|s_0)} s_1 \xrightarrow{a \sim \pi_\theta(\cdot|s_1)} s_2 \dots s_{L-1} \xrightarrow{a \sim \pi_\theta(\cdot|s_{L-1})} s_L \in S^*.$$

Series of actions recorded along the trajectory allows us to easily construct the nested sequence $\mathbb{S}_L \subseteq \dots \subseteq \mathbb{S}_1 \subseteq \mathbb{S}_0 = [n]$.

$R(s^*)$ is the reward function that specifies the reward after reaching a terminal state s^* . The reward function is defined as negative of the MMF reconstruction loss such that

$$R(s^*) = -\|\mathbf{A} - \mathbf{U}_1^T \dots \mathbf{U}_L^T \mathbf{H} \mathbf{U}_L \dots \mathbf{U}_1\|_F. \quad (6.21)$$

We want to maximize this final reward that is equivalent to minimize error of MMF in Frobenius norm (as in problem (6.8)). Evaluation of the reward requires the Stiefel manifold optimization (see Section 6.5) for rotations $\{\mathbf{U}_\ell\}_{\ell=1}^L$. Obviously, the final reward in Eq. (6.21) is the most important. However, to improve the training quality of the policy, we can define the intermediate reward $R(s)$ for non-terminal states $s = (\mathbf{A}_{\mathbb{S}, \mathbb{S}}, \mathbb{S}, \ell) \notin S^*$ as the immediate improvement of the ℓ -th resolution ($L > \ell > 0$):

$$R(s) = -\|[\mathbf{U}_\ell \mathbf{A}_{\ell-1} \mathbf{U}_\ell^T]_{\mathbb{S}, \mathbb{S}}\|_{\text{resi}}^2. \quad (6.22)$$

Along the trajectory (s_0, s_1, \dots, s_L) , we generate the corresponding sequence of rewards (r_1, r_2, \dots, r_L) based on (6.21, 6.22).

γ is the discount factor, a penalty to uncertainty of future rewards, $0 < \gamma \leq 1$. We define the return or discounted future reward g_ℓ for $\ell = 0, \dots, L - 1$ as

$$g_\ell = \sum_{k=0}^{L-\ell-1} \gamma^k r_{\ell+k+1}, \quad (6.23)$$

which in the case of $\gamma = 1$ indicates simply accumulating all the immediate rewards and the final reward along the trajectory.

6.6.3 Policy gradient methods

Policy gradient has been a widely used approach to solve reinforcement learning problems that targets at modeling and optimizing the policy directly (Sutton and Barto, 2018). Monte-Carlo policy gradient (REINFORCE) (Williams, 1988) (Williams, 1992) (Sutton et al., 2000) depends on an estimated return by Monte-Carlo methods using episode samples to update the learnable parameters θ of the policy network π_θ . We define the value of state s when we follow a policy π as $V^\pi(s) = \mathbb{E}_{a \sim \pi}[g_\ell | s_\ell = s]$. The value of (state, action) pair when we follow a policy π is defined similarly as $Q^\pi(s, a) = \mathbb{E}_{a \sim \pi}[g_\ell | s_\ell = s, a_\ell = a]$. The value of the reward objective function depends on the policy and is defined as

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s, a) \quad (6.24)$$

where $d^\pi(s) = p(s_L = s | s_0, \pi_\theta)$ is the stationary distribution of Markov chain for π_θ . It is important to remark that our MDP process terminates after a finite number of transitions (e.g., L), so $d^\pi(s)$ is the probability that we end up at state s when starting from s_0 and following policy π_θ for L steps. (Sutton et al., 2000) has shown that an unbiased estimate of

the gradient of (6.24) can be obtained from experience using an approximate value function. The expectation of the sample gradient is equal to the actual gradient:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi}[Q^{\pi}(s, a)\nabla_{\theta} \log \pi_{\theta}(a|s)] = \mathbb{E}_{\pi}[g_{\ell}\nabla_{\theta} \log \pi_{\theta}(a_{\ell}|s_{\ell})] \quad (6.25)$$

that allows us to update our policy gradient by measuring g_{ℓ} from real sample trajectories. Based on (6.25), the update rule for policy parameters is simply as

$$\theta \leftarrow \theta + \eta \gamma^{\ell} g_{\ell} \nabla_{\theta} \log \pi_{\theta}(a_{\ell}|s_{\ell}) \quad \text{for } \ell = 0, \dots, L-1;$$

where η is the learning rate, that is used in training our policy network π_{θ} .

6.6.4 Graph convolutional policy network

In this section, we design our policy network π_{θ} as a graph neural network (GNN) with the message passing scheme. We consider the symmetric matrix \mathbf{A} being represented by a weighted undirected graph $\mathcal{G} = (V, E)$ in which V is the set of nodes such that each node corresponds to a row/column of \mathbf{A} , and E is the set of edges such that the edge $(i, j) \in E$ has the weight $\mathbf{A}_{i,j}$. As defined in Section 6.6.2, a state $s \in S$ is a tuple $(\mathbf{A}_{\mathbb{S}, \mathbb{S}}, \mathbb{S}, \ell)$ in which $\mathbf{A}_{\mathbb{S}, \mathbb{S}}$ is the sub-matrix restricted to the active rows/columns \mathbb{S} , and an action $a \in A$ is a tuple (\mathbb{I}, \mathbb{T}) in which \mathbb{I} is the set of k indices corresponding to the $(\ell+1)$ -th rotation and \mathbb{T} is the set of indices to spit out as wavelets. Practically, a state can be simply represented by a single binary vector such that if a bit is 1 then the corresponding index is active, without the need to explicitly storage matrix $\mathbf{A}_{\mathbb{S}, \mathbb{S}}$ that can be efficiently constructed from \mathbf{A} by any numerical toolkit. Our GNN policy network $\pi_{\theta}(a|s)$ learns to encode the underlying graph represented by $\mathbf{A}_{\mathbb{S}, \mathbb{S}}$ and returns a sample of valid action such that $\mathbb{T} \subset \mathbb{I} \subset \mathbb{S}$. In Section 6.6.1, we assume that \mathbb{T} contains only a single index that we will call as the *pivot* i^* (e.g., $\mathbb{T} = \{i^*\}$). Thus, the task of our GNN model is to learn to select the pivot i^* first, and then

select the rest $k - 1$ indices of \mathbb{I} that are highly correlated to the pivot.

The simplest implementation of GNNs is Message Passing Neural Networks (MPNNs) (Gilmer et al., 2017). Suppose that the node embeddings (messages) $\mathbf{M}_0 \in \mathbb{R}^{N \times F}$ are initialized by the input node features where N is the number of nodes and F is the number of features for each node. Iteratively, the messages are propagated from each node to its neighborhood, and then transformed by a combination of linear transformations and non-linearities, e.g.,

$$\hat{\mathbf{M}}_t = \mathbf{A}\mathbf{M}_{t-1}, \quad \mathbf{M}_t = \sigma(\hat{\mathbf{M}}_t \mathbf{W}_{t-1}), \quad (6.26)$$

where \mathbf{A} is the adjacency matrix; $\hat{\mathbf{M}}_t$ and $\mathbf{M}_t \in \mathbb{R}^{N \times D}$ are the aggregated messages (by summing over each node's neighborhood) and the output messages at the t 'th iteration, respectively; σ is a element-wise non-linearity function (e.g., sigmoid, ReLU, etc.); and \mathbf{W} s are learnable weight matrices such that $\mathbf{W}_0 \in \mathbb{R}^{F \times D}$ and $\mathbf{W}_t \in \mathbb{R}^{D \times D}$ for $t > 0$. Basically, the set of learnable parameters θ of our policy network π includes all \mathbf{W} s. In some cases, a graph Laplacian \mathbf{L} is used instead of the adjacency matrix \mathbf{A} in model (6.26), for example, graph Laplacian $\mathbf{L} = \mathbf{D}^{-1}\mathbf{A}$ or its symmetric normalized version $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. One way to incorporate the set of active rows/columns/nodes \mathbb{S} into our GNN model is by initializing the input node feature with a binary label ($f = 1$) such that a node v has label 1 if the v -th row/column is still active, otherwise 0. For a more efficient implementation, we can execute the message passing in the block $\mathbf{A}_{\mathbb{S}, \mathbb{S}}$ only. Supposing that the message passing scheme is executed for T iterations, we concatenate messages from every iteration together into the final embedding:

$$\mathbf{M} = \bigoplus_{t=1}^T \mathbf{M}_t \in \mathbb{R}^{N \times DT}. \quad (6.27)$$

Model (6.26) produces the embedding for each node that allows us to define a sampling procedure to select the pivot i^* . Given the final embedding \mathbf{M} from Eq. (6.27), we define

the probability \mathbf{P}_i that node $i \in \mathbb{S}$ is being selected as the pivot as:

$$P_i = \frac{\exp(\hat{P}_i)}{\sum_{j \in \mathbb{S}} \exp(\hat{P}_j)}, \quad \text{where} \quad \hat{P}_i = \sum_f \mathbf{M}_{i,f}.$$

In order to make the sampling procedure differentiable for backpropagation, we apply the Gumbel-max trick (Gumbel, 1954) (Maddison et al., 2014) (Jang et al., 2017) that provides a simple and efficient way to draw sample i^* as follows:

$$i^* = \text{one-hot}\left(\arg \max_{i \in \mathbb{S}} [G_i + \log P_i]\right),$$

where G_i are i.i.d samples drawn from $\text{Gumbel}(0, 1)$. Technically, the sample is represented by a one-hot vector such that the i^* -th element is 1. Similarly, \mathbb{T} , \mathbb{I} and \mathbb{S} are represented by vectors in $\{0, 1\}^N$ in which a 1-element indicates the existence of the corresponding index in the set. Furthermore, the set union and minus operations (e.g., $\mathbb{S} \setminus \mathbb{T}$) can be easily done by vector addition and subtraction, respectively.

Given the pivot i^* , we compute the similarity score between i^* and other nodes $i \in \mathbb{S}$ as $C_i = \langle \mathbf{M}_{i^*, :}, \mathbf{M}_{i,:} \rangle$. Finally, we sample $k - 1$ nodes with the highest similarity scores to i sequentially (one-by-one) without replacement by the Gumbel-max trick, that completes our sampling procedure for action $a = (\mathbb{I}, \mathbb{T})$.

The REINFORCE (Williams, 1988) (Williams, 1992) (Sutton et al., 2000) update rule for policy parameters is

$$\theta \leftarrow \theta + \eta \gamma^\ell g_\ell \nabla_\theta \log \pi_\theta(a_\ell | s_\ell) \quad \text{for } \ell = 0, \dots, L - 1; \quad (6.28)$$

where η is the learning rate, that is used in training our policy network π_θ in Algorithm 17.

6.6.5 The learning algorithm

Putting everything together, our MMF learning algorithm is sketched in Algorithm 17. Iteratively: (1) we sample a trajectory by running the policy network that indicates the indices for rotation and wavelet for each resolution, (2) we apply the Stiefel manifold optimization to find the rotations, and (3) we compute the future rewards and update the parameters of the policy network by REINFORCE accordingly. The learning terminates when the average error over a window of size ω iterations increases.

Algorithm 17 MMF learning algorithm optimizing problem (6.8)

```

1: Input: Matrix  $\mathbf{A}$ , number of resolutions  $L$ , and constants  $k, \gamma, \eta$ , and  $\omega$ .
2: Initialize the policy parameter  $\theta$  at random.
3: while true do
4:   Start from state  $s_0$                                       $\triangleright s_0 \triangleq (\mathbf{A}, [n], 0)$ 
5:   Initialize  $\mathbb{S}_0 \leftarrow [n]$                                  $\triangleright$  All rows/columns are active at the beginning.
6:   for  $\ell = 0, \dots, L - 1$  do
7:     Sample action  $a_\ell = (\mathbb{I}_{\ell+1}, \mathbb{T}_{\ell+1})$  from  $\pi_\theta(a_\ell|s_\ell)$ .       $\triangleright$  See Section 6.6.4.
8:      $\mathbb{S}_{\ell+1} \leftarrow \mathbb{S}_\ell \setminus \mathbb{T}_{\ell+1}$                        $\triangleright$  Eliminate the wavelet index (indices).
9:      $s_{\ell+1} \leftarrow (\mathbf{A}_{\mathbb{S}_{\ell+1}, \mathbb{S}_{\ell+1}}, \mathbb{S}_{\ell+1}, \ell + 1)$      $\triangleright$  New state with a smaller active set.
10:    end for
11:    Given  $\{\mathbb{I}_\ell\}_{\ell=1}^L$ , minimize objective (6.13) by Stiefel manifold optimization to find
12:       $\{\mathbf{O}_\ell\}_{\ell=1}^L$ .                                               $\triangleright \mathbf{U}_\ell = \mathbf{I}_{n-k} \oplus \mathbb{I}_\ell \mathbf{O}_\ell$ 
13:    for  $\ell = 0, \dots, L - 1$  do
14:      Estimate the return  $g_\ell$  based on Eq. (6.21), Eq. (6.22), and Eq. (6.23).
15:       $\theta \leftarrow \theta + \eta \gamma^\ell g_\ell \nabla_\theta \log \pi_\theta(a_\ell|s_\ell)$        $\triangleright$  REINFORCE policy update in Eq. (6.28)
16:    end for
17:  Terminate if the average error of the last  $\omega$  iterations increases.     $\triangleright$  Early stopping.
18: end while

```

6.6.6 2-phase process & Transfer learning

The learning algorithm is expensive due to the Stiefel manifold optimization in line 11 to find the optimal rotations \mathbf{O}_ℓ that are used to compute the rewards g_ℓ . In practice, we propose a 2-phase process that is more efficient:

- **Phase 1:** Reinforcement learning to find the sequence of indices, but instead of man-

ifold optimization, we just use the closed-form solutions for \mathbf{O}_ℓ as the eigenvectors of $\mathbf{A}_{\mathbb{S}_\ell,:}\mathbf{A}_{\mathbb{S}_\ell,:}^T$ to estimate the rewards. In all our experiments, we implement the policy network by two graph neural networks, one to select the pivot (wavelet index) and the another one to select $K - 1$ indices, with 4 layers of message passing and hidden dimension of 10. The input node feature for node v (or the v -th row) is binary: 1 if $v \in \mathbb{S}_\ell$, otherwise 0. We use $\gamma = 1$ as the discount factor and learning rate $\eta = 10^{-3}$.

- **Phase 2:** Given a sequence of indices found by the previous phase, we apply Stiefel manifold optimization to actually find the optimal rotations accordingly.

Ideally, we want our policy network π_θ to be *universal* in the sense that the same trained policy can be applied to different graphs with little adaptation or without any further training. However, the search space is gigantic with large graphs such as social networks, and the cost of training the policy is computationally expensive. Therefore, we apply the idea of *transfer learning* that is to reuse or transfer information from previously learned tasks (source tasks) into new tasks (target tasks). The source task here is to train our GNN policy on a dataset of small graphs (e.g., possibly synthetic graphs) that are much faster to train on, and the target task is to run the trained policy on the large actual graph. For example, given citation networks with thousands of nodes such as Cora and Citeseer (Sen et al., 2008), we generate the dataset for policy training by partitioning the big graph into many smaller connected clusters. The learning algorithm can be easily modified for such a purpose (e.g., training multiple graphs simultaneously).

6.7 Wavelet Networks on Graphs

6.7.1 Motivation

The eigendecomposition of the normalized graph Laplacian operator $\tilde{\mathbf{L}} = \mathbf{U}^T \mathbf{H} \mathbf{U}$ can be used as the basis of a graph Fourier transform. (Shuman et al., 2013) defines graph Fourier

transform (GFT) on a graph $\mathcal{G} = (V, E)$ of a graph signal $\mathbf{f} \in \mathbb{R}^n$ (that is understood as a function $f : V \rightarrow \mathbb{R}$ defined on the vertices of the graph) as $\hat{\mathbf{f}} = \mathbf{U}^T \mathbf{f}$, and the inverse graph Fourier transform as $\mathbf{f} = \mathbf{U} \hat{\mathbf{f}}$. Analogously to the classical Fourier transform, GFT provides a way to represent a graph signal in two domains: the vertex domain and the graph spectral domain; to filter graph signal according to smoothness; and to define the graph convolution operator, denoted as $*_{\mathcal{G}}$:

$$\mathbf{f} *_{\mathcal{G}} \mathbf{g} = \mathbf{U}((\mathbf{U}^T \mathbf{g}) \odot (\mathbf{U}^T \mathbf{f})), \quad (6.29)$$

where \mathbf{g} denotes the convolution kernel, and \odot is the element-wise Hadamard product. If we replace the vector $\mathbf{U}^T \mathbf{g}$ by a diagonal matrix $\tilde{\mathbf{g}}$, then we can rewrite the Hadamard product in Eq. (6.29) to matrix multiplication as $\mathbf{U} \tilde{\mathbf{g}} \mathbf{U}^T \mathbf{f}$ (that is understood as filtering the signal \mathbf{f} by the filter $\tilde{\mathbf{g}}$). Based on GFT, (Bruna et al., 2014) and (Defferrard et al., 2016) construct convolutional neural networks (CNNs) learning on spectral domain for discrete structures such as graphs. However, there are two fundamental limitations of GFT:

- High computational cost: eigendecomposition of the graph Laplacian has complexity $O(n^3)$, and “Fourier transform” itself involves multiplying the signal with a dense matrix of eigenvectors.
- The graph convolution is not localized in the vertex domain, even if the graph itself has well defined local communities.

To address these limitations, we propose a modified spectral graph network based on the MMF wavelet basis rather than the eigenbasis of the Laplacian. This has the following advantages: (i) the wavelets are generally localized in both vertex domain and frequency, (ii) the individual basis transforms are sparse, and (iii) MMF provides a computationally efficient way of decomposing graph signals into components at different granularity levels and an excellent basis for sparse approximations.

6.7.2 Network construction

In the case \mathbf{A} is the normalized graph Laplacian of a graph $\mathcal{G} = (V, E)$, the wavelet transform (up to level L) expresses a graph signal (function over the vertex domain) $f : V \rightarrow \mathbb{R}$, without loss of generality $f \in \mathbb{V}_0$, as:

$$f(v) = \sum_{\ell=1}^L \sum_m \alpha_m^\ell \psi_m^\ell(v) + \sum_m \beta_m \phi_m^L(v), \quad \text{for each } v \in V,$$

where $\alpha_m^\ell = \langle f, \psi_m^\ell \rangle$ and $\beta_m = \langle f, \phi_m^L \rangle$ are the wavelet coefficients. At each level, a set of coordinates $\mathbb{T}_\ell \subset \mathbb{S}_{\ell-1}$ are selected to be the wavelet indices, and then to be eliminated from the active set by setting $\mathbb{S}_\ell = \mathbb{S}_{\ell-1} \setminus \mathbb{T}_\ell$ (see Section 6.6.1). Practically, we make the assumption that we only select 1 wavelet index for each level (see Section 6.6.1) that results in a single mother wavelet $\psi^\ell = [\mathbf{A}_\ell]_{i^*,:}$ where i^* is the selected index (see Section 6.6.4). We get exactly L mother wavelets $\bar{\psi} = \{\psi^1, \psi^2, \dots, \psi^L\}$. On the other hand, the active rows of $\mathbf{H} = \mathbf{A}_L$ make exactly $N - L$ father wavelets $\bar{\phi} = \{\phi_m^L = \mathbf{H}_{m,:}\}_{m \in \mathbb{S}_L}$. In total, a graph of N vertices has exactly N wavelets (both mothers and fathers). Analogous to the convolution based on GFT (Bruna et al., 2014), each convolution layer $k = 1, \dots, K$ of our wavelet network transforms an input vector $\mathbf{f}^{(k-1)}$ of size $|V| \times F_{k-1}$ into an output $\mathbf{f}^{(k)}$ of size $|V| \times F_k$ as

$$\mathbf{f}_{:,j}^{(k)} = \sigma \left(\mathbf{W} \sum_{i=1}^{F_{k-1}} \mathbf{g}_{i,j}^{(k)} \mathbf{W}^T \mathbf{f}_{:,i}^{(k-1)} \right) \quad \text{for } j = 1, \dots, F_k, \quad (6.30)$$

where \mathbf{W} is our wavelet basis matrix as we concatenate $\bar{\phi}$ and $\bar{\psi}$ column-by-column, $\mathbf{g}_{i,j}^{(k)}$ is a parameter/filter in the form of a diagonal matrix learned in spectral domain, and σ is an element-wise linearity (e.g., ReLU, sigmoid, etc.).

For example, in node classification tasks, assume the number of classes is C , the set of labeled nodes is V_{label} , and we are given a normalized graph Laplacian $\tilde{\mathbf{L}}$ and an input node

feature matrix $\mathbf{f}^{(0)}$. First of all, we apply our MMF learning algorithm 17 to factorize \tilde{L} and produce our wavelet basis matrix \mathbf{W} . Then, we construct our wavelet network as a multi-layer CNNs with each convolution is defined as in Eq. (6.30) that transforms $\mathbf{f}^{(0)}$ into $\mathbf{f}^{(K)}$ after K layers. The top convolution layer K -th returns exactly $F_K = C$ features and uses softmax instead of the nonlinearity σ for each node. The loss is the cross-entropy error over all labeled nodes as:

$$\mathcal{L} = - \sum_{v \in V_{\text{label}}} \sum_{c=1}^C \mathbf{y}_{v,c} \ln \mathbf{f}_{v,c}^{(K)}, \quad (6.31)$$

where $\mathbf{y}_{v,c}$ is a binary indicator that is equal to 1 if node v is labeled with class c , and 0 otherwise. The set of weights $\{\mathbf{g}^{(k)}\}_{k=1}^K$ are trained using gradient descent optimizing the loss in Eq. (6.31).

6.8 Experiments

6.8.1 Matrix factorization

We evaluate the performance of our MMF learning algorithm in comparison with the original greedy algorithm (Kondor et al., 2014) and the Nyström method (Gittens and Mahoney, 2013) in the task of matrix factorization on 3 datasets: (i) normalized graph Laplacian of the Karate club network ($N = 34$, $E = 78$) (Zachary, 1976); (ii) a Kronecker product matrix ($N = 512$), \mathcal{K}_1^n , of order $n = 9$, where $\mathcal{K}_1 = ((0, 1), (1, 1))$ is a 2×2 seed matrix (Leskovec et al., 2010); and (iii) normalized graph Laplacian of a Cayley tree or Bethe lattice with coordination number $z = 4$ and 4 levels of depth ($N = 161$). The rotation matrix size K are 8, 16 and 8 for Karate, Kronecker and Cayley, respectively. Meanwhile, the original greedy MMF is limited to $K = 2$ and implements an exhaustive search to find an optimal pair of indices for each rotation. For both versions of MMF, we drop $c = 1$ columns after each rotation, which results in a final core size of $d_L = N - c \times L$. The exception is for the Kronecker matrix ($N = 512$), our learning algorithm drops up to 8 columns (for example,

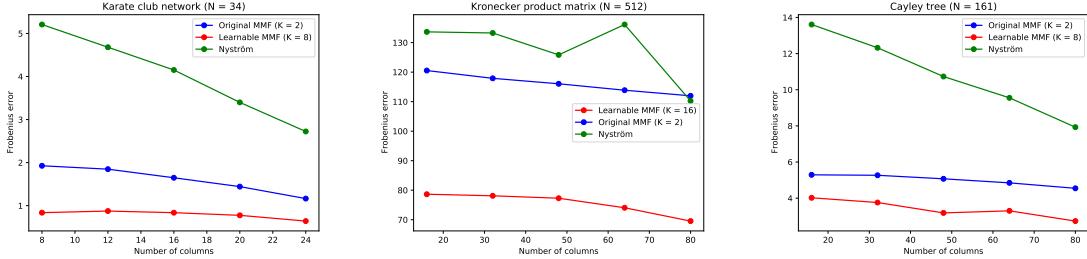


Figure 6.5: Matrix factorization for the Karate network (left), Kronecker matrix (middle), and Cayley tree (right). Our learnable MMF consistently outperforms the classic greed methods.

$L = 62$ and $c = 8$ results into $d_L = 16$) to make sure that the number of learnable parameters $L \times K^2$ is much smaller the matrix size N^2 . Our learning algorithm compresses the Kronecker matrix down to 6–7% of its original size. The details of efficient training reinforcement learning with the policy networks implemented by GNNs are included in the Appendix.

For the baseline of Nyström method, we randomly select, by uniform sampling without replacement, the same number d_L columns \mathbf{C} from \mathbf{A} and take out \mathbf{W} as the corresponding $d_L \times d_L$ submatrix of \mathbf{A} . The Nyström method approximates $\mathbf{A} \approx \mathbf{C}\mathbf{W}^\dagger\mathbf{C}^T$. We measure the approximation error in Frobenius norm. Figure 6.5 shows our MMF learning algorithm consistently outperforms the original greedy algorithm and the Nyström baseline given the same number of active columns, d_L . Figure 6.6 depicts the wavelet bases at different levels of resolution.

6.8.2 Node classification on citation graphs

To evaluate the wavelet bases returned by our learnable MMF algorithm, we construct our wavelet networks (WNNs) as in Sec. 6.7 and apply it to the task of node classification on two citation graphs, Cora ($N = 2,708$) and Citeseer ($N = 3,312$) (Sen et al., 2008) in which nodes and edges represent documents and citation links. Each document in Cora and Citeseer has

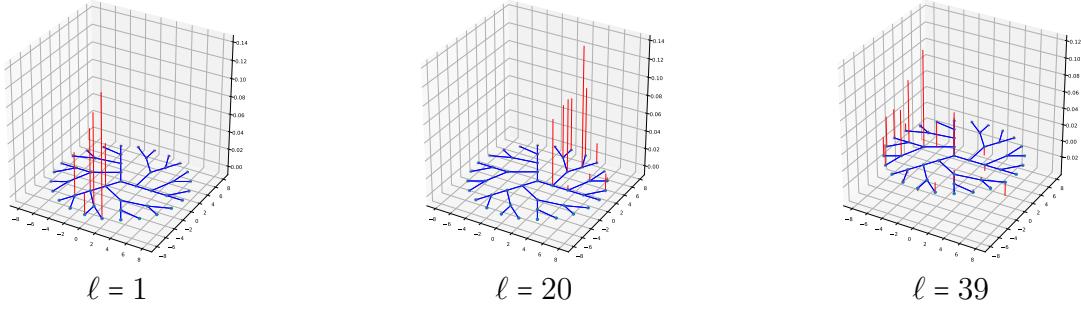


Figure 6.6: Visualization of some of the wavelets on the Cayley tree of 46 vertices. The low index wavelets (low ℓ) are highly localized, whereas the high index ones are smoother and spread out over large parts of the graph.

an associated feature vector (of length 1,433 resp. 3,703) computed from word frequencies, and is classified into one of 7 and 6 classes, respectively. We factorize the normalized graph Laplacian by learnable MMF with $K = 16$ to obtain the wavelet bases. The resulting MMF wavelets are sparse, which makes it possible to run a fast transform on the node features by sparse matrix multiplication: only 4.69% and 15.25% of elements are non-zero in Cite-seer and Cora, respectively. In contrast, Fourier bases given by eigendecomposition of the graph Laplacian are completely dense (100% of elements are non-zero). We evaluate our WNNs with 3 different random splits of train/validation/test: (1) 20%/20%/60% denoted as MMF_1 , (2) 40%/20%/40% denoted as MMF_2 , and (3) 60%/20%/20% denoted as MMF_3 . The WNN learns to encode the whole graph with 6 layers of spectral convolution and 100 hidden dimensions for each node. During training, the network is only trained to predict the node labels in the training set. Hyperparameter searching is done on the validation set. The number of epochs is 256 and we use the Adam optimization method (Kingma and Ba, 2015) with learning rate $\eta = 10^{-3}$. We report the final test accuracy for each split in Table 6.1.

We compare with several traditional methods and deep learning methods including other spectral graph convolution networks such as Spectral CNN, and graph wavelet neural net-

Table 6.1: Node classification on citation graphs. Baseline results are taken from (Xu et al., 2019).

Method	Cora	Citeseer
MLP	55.1%	46.5%
ManiReg (Belkin et al., 2006)	59.5%	60.1%
SemiEmb (Weston et al., 2008)	59.0%	59.6%
LP (Zhu et al., 2003)	68.0%	45.3%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%
ICA (Getoor, 2005)	75.1%	69.1%
Planetoid (Yang et al., 2016)	75.7%	64.7%
Spectral CNN (Bruna et al., 2014)	73.3%	58.9%
ChebyNet (Defferrard et al., 2016)	81.2%	69.8%
GCN (Kipf and Welling, 2017)	81.5%	70.3%
MoNet (Monti et al., 2017)	81.7%	N/A
GWNN (Xu et al., 2019)	82.8%	71.7%
MMF₁	84.35%	68.07%
MMF₂	84.55%	72.76%
MMF₃	87.59%	72.90%

works (GWNN). Baseline results are taken from (Xu et al., 2019). Our wavelet networks perform competitively against state-of-the-art methods in the field.

6.8.3 Graph classification

We also tested our WNNs on standard graph classification benchmarks including four bioinformatics datasets: (1) MUTAG, which is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds with 7 discrete labels (Debnath et al., 1991); (2) PTC, which consists of 344 chemical compounds with 19 discrete labels that have been tested for positive or negative toxicity in lab rats (Toivonen et al., 2003); (3) PROTEINS, which contains 1,113 molecular graphs with binary labels, where nodes are secondary structure elements (SSEs) and there is an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space (Borgwardt et al., 2005); (4) NCI1, which has 4,110 compounds with binary labels, each screened for activity against small cell lung cancer and ovarian cancer lines (Wale

et al., 2008). Each molecule is represented by an adjacency matrix, and we represent each atomic type as a one-hot vector and use them as the node features.

We factorize all normalized graph Laplacian matrices in these datasets by MMF with $K = 2$ to obtain the wavelet bases. Again, MMF wavelets are **sparse** and suitable for fast transform via sparse matrix multiplication, with the following average percentages of non-zero elements for each dataset: 19.23% (MUTAG), 18.18% (PTC), 2.26% (PROTEINS) and 11.43% (NCI1).

Our WNNs contain 6 layers of spectral convolution, 32 hidden units for each node, and are trained with 256 epochs by Adam optimization with an initial learning rate of 10^{-3} . We follow the evaluation protocol of 10-fold cross-validation from (Zhang et al., 2018). We compare our results to several deep learning methods and popular graph kernel methods. Baseline results are taken from (Maron et al., 2019b). Our WNNs outperform 7/8, 7/8, 8/8, and 2/8 baseline methods on MUTAG, PTC, PROTEINS, and NCI1, respectively (see Table 6.2).

Table 6.2: Graph classification. Baseline results are taken from (Maron et al., 2019b).

Method	MUTAG	PTC	PROTEINS	NCI1
DGCNN (Zhang et al., 2018)	85.83 ± 1.7	58.59 ± 2.5	75.54 ± 0.9	74.44 ± 0.5
PSCN (Niepert et al., 2016)	88.95 ± 4.4	62.29 ± 5.7	75 ± 2.5	76.34 ± 1.7
DCNN (Atwood and Towsley, 2016)	N/A	N/A	61.29 ± 1.6	56.61 ± 1.0
CCN (Kondor et al., 2018a)	91.64 ± 7.2	70.62 ± 7.0	N/A	76.27 ± 4.1
GK (Shervashidze et al., 2009)	81.39 ± 1.7	55.65 ± 0.5	71.39 ± 0.3	62.49 ± 0.3
RW (Vishwanathan et al., 2010)	79.17 ± 2.1	55.91 ± 0.3	59.57 ± 0.1	N/A
PK (Neumann et al., 2016)	76 ± 2.7	59.5 ± 2.4	73.68 ± 0.7	82.54 ± 0.5
WL (Shervashidze et al., 2011)	84.11 ± 1.9	57.97 ± 2.5	74.68 ± 0.5	84.46 ± 0.5
IEGN (Maron et al., 2019b)	84.61 ± 10	59.47 ± 7.3	75.19 ± 4.3	73.71 ± 2.6
MMF	86.31 ± 9.47	67.99 ± 8.55	78.72 ± 2.53	71.04 ± 1.53

6.9 Software

We implemented our learning algorithm for MMF and the wavelet networks by PyTorch deep learning framework (Paszke et al., 2019). We released our implementation at

[https://github.com/risilab/Learnable_MMF/.](https://github.com/risilab/Learnable_MMF/)

6.10 Chapter Conclusion

In this chapter, we introduced a general algorithm based on reinforcement learning and Stiefel manifold optimization to optimize Multiresolution Matrix Factorization (MMF). We find that the resulting learnable MMF consistently outperforms the existing greedy and heuristic MMF algorithms in factorizing and approximating hierarchical matrices. Based on the wavelet basis returned from our learning algorithm, we define a corresponding notion of spectral convolution and construct a wavelet neural network for graph learning problems. Thanks to the sparsity of the MMF wavelets, the wavelet network can be efficiently implemented with sparse matrix multiplication. We find that this combination of learnable MMF factorization and spectral wavelet network yields state of the art results on standard node classification and molecular graph classification.

CHAPTER 7

FUTURE WORKS

For future research directions, the thesis can be extended and further developed into the following projects with the aim for drug and material discovery:

- Theoretical investigation of symmetry-preserving and group equivariant neural models with higher order interactions on hypergraphs.
- Large-scale implementation of rotationally equivariant N-Body networks.
- Learning to model large hierarchical structures such as proteins by multiresolution equivariant graph models.
- Learning to capture diffraction symmetries by permutation and rotation equivariant models in order to generate highly symmetric structures such as crystal structures in X-ray crystallography.
- Molecule optimization by deep generative models on graphs to generate new drugs.
- Large-scale implementation of learnable multiresolution matrix/tensor factorization and its sparse wavelet transform on graphs with applications in hierarchical data visualization.

7.1 Symmetry-preserving and equivariant models with higher order interactions: more theoretical understanding

Existing graph models have been built based on the pairwise interaction paradigm (e.g., information sharing via edges), which alone cannot capture the higher order interactions in many complex systems (e.g., n-fold atomic interactions in N-Body systems) and multirelational data that are naturally represented by hypergraphs. The aim of this project is to model and analyze hypergraph neural network architecture that is fully equivariant with respect to transformations under the permutation group. My plan is to construct and compare

the representative power of three forms of hypergraph convolution: (i) spectral convolution via hypergraph Fourier transform (Zhang et al., 2019), (ii) simplicial/spectral convolution with the basis of transformation given by eigenvectors of the combinatorial Laplace operator on simplicial complexes (Horak and Jost, 2013), and (iii) topological convolution (by message passing) defined on topological spaces such as simplicial complexes of graphs (Jonsson, 2008). One of the applications is to model the n-fold atomic interactions in molecular graphs and N-Body systems.

7.2 Large-scale implementation of the generalized N-Body networks

Learning the behavior and properties of complex many-body physical systems is a challenging task due to the fact that the model must be invariant with respect to rotation and translation. Previous works attempting to model the N-Body systems by GNNs such as the Interaction Networks (Battaglia et al., 2016) have two fundamental limitations: (i) GNNs itself is limited to the pairwise paradigm, and (ii) the model does not preserve the rotational symmetry. (Kondor, 2018) schemed the abstract definition and construction of an N-Body network architecture that incorporates both higher interactions among particles while respecting rotational and translational symmetries. My previous work, Cormorant (Anderson et al., 2019), partially realized the vision of (Kondor, 2018) but still is limited to the pairwise paradigm. My plan is to extend Cormorant by generalizing the topological convolution defined on the simplicial complexes (e.g., 0-simplices are atoms, 1-simplices are bonds, 2-simplices are triples of atoms, etc.). Since a layer of the network requires a large number of convolutional operations, I plan to create an efficient and parallel implementation in C++/CUDA to execute serials of Clebsch-Gordan transforms on tensor products of multiple spherical tensors simultaneously.

7.2.1 Application in LHC particle tracking & particle discovery

The Large Hadron Collider (LHC) is the largest high-energy particle collider ever constructed to probe fundamental physics questions (Evans and Bryant, 2008). ATLAS and CMS (Collaboration et al., 2008) experiments measure the energy and momentum of the particles escaping from the collision of two counter-circulating proton beams in the center of the detectors. It is important to track and reconstruct the trajectories of charged particles to identify particles and measure their charge and momentum. Traditional tracking algorithms such as Combinatorial Kalman Filter (CKF) (Braun, 2019) scale quadratically or worse, and require an intensive amount of CPU computation. (Tsaris et al., 2018) preliminarily propose the use of advanced machine learning algorithms that scale linearly and computationally feasible to the LHC track reconstruction problem. However, the fundamental limit of (Tsaris et al., 2018) is its lack of a 3D geometric model to represent the geometry information including particles' positions and momentums, (detector modules) sensors' locations and orientations. Therefore, I propose a combination of time-series/sequence predicting models such as LSTM (Hochreiter and Schmidhuber, 1997) networks and equivariant models such as Lorentz group (Bogatskiy et al., 2020) or Cormorant to predict the probabilities of assigning hits to tracks. My plan is to model prototype architectures, and experiment on The European Organization for Nuclear Research (CERN)'s particle tracking datasets in comparision with traditional tracking methods.

Furthermore, symmetry-preserving neural networks in my prior works can potentially be applied into classifying particles and discovering new ones. Given the large amount of data collected from particle collisions, it is a necessity to develop efficient computing infrastructure that allows us to train and evaluate our equivariant neural networks¹. This is an exciting

1. Equivariant neural networks that exploit the symmetry of data will require much less training samples and data augmentation in comparison with non-equivariant models.

interdisciplinary research direction to bring machine learning, artificial intelligence and computer science into further advancing our understanding of the fundamental blocks of the Universe.

7.3 Learning and generating hierarchical multiscale structures and highly symmetric structures by deep generative models

7.3.1 Modeling and generating proteins

Several deep learning approaches have been applied to modeling and generating proteins in recent years. The most prominent approach is language modeling borrowed from the field of Natural Language Processing (NLP) to model the primary structure of proteins (i.e. the sequence of amino acids) (O'Connell et al., 2018) (Bepler and Berger, 2019) (Riesselman et al., 2019) (Alley et al., 2019) (Heinzinger et al., 2019) (Ingraham et al., 2019). However, proteins are naturally hierarchical structures and language modeling such as Transformer or AlphaFold is computationally expensive. Therefore, I propose the use of learning to cluster algorithm as in our MGVAE (Hy and Kondor, 2021b) to coarsen the primary structure, and introduce 3D equivariant Transformer model built based on our Cormorant (Anderson et al., 2019) that incorporates both graph and 3D information to learn on the coarsened structures. I plan to implement this complex neural network model and tackle the following two major tasks in protein modelling:

1. *Function prediction:* This is a supervised learning problem in which the amino-acid sequence and/or the 3D structure is known and the functionality is needed as output of a neural network. The property to predict can either be a protein-level property, such as a classification as an enzyme or non-enzyme, or a residue-level property, such as the sites or motifs of phosphorylation and cleavage by proteases (see (Gao et al., 2020) for a comprehensive survey). The advantage of my proposal is the adaptive/learnable

multiscale representation of the protein to capture multiple granularity levels that is potentially suitable for many different tasks (i.e. one model to solve all problems).

2. *Protein design*: The objective is to obtain a novel protein sequence that will fold into a desired structure or perform a specific function, such as catalysis. I plan to follow the deep generative models approach, in particular variational autoencoders (VAEs). In this case, the encoder learns to coarsen the protein into multiple resolutions (e.g., the base resolution is the primary structure) while maps each resolution into a continuous latent space. Different from the conventional VAE, the encoder outputs a hierarchy of latents such that the decoder then reconstructs each latent into its corresponding structure. This hierarchical VAE can be jointly trained with properties prediction to construct a latent space (representation) correlated with the properties. The resulting continuous real-valued representation can then be used to generate new sequences likely to have the desired properties.

7.3.2 *Classifying and generating crystalized structures*

Highly symmetric objects such as crystals play a crucial role in materials science. X-ray crystallography has been widely used as a tool to determine the atomic and molecular structure of a crystal. Knowledge of crystal structure, the way atoms are arranged in space, is important for predicting properties of a material. Recently, convolutional neural networks (CNNs) has been applied to classify crystal structures (Ziletti et al., 2018) and to decode crystallography (Aguiar et al., 2019) based on the diffraction fingerprint and high-resolution electron imaging. However, CNNs are only invariant to translation, but not to permutation and rotation, thus this conventional approach is not suitable. To address this limitation, I propose the use of a mixture of permutation and rotation equivariant models to capture the diffraction symmetry. I predict equivariant models would require significantly less amount of training data and more robust against data noises. Given this powerful equivariant model,

I plan to develop constrained generative models to generate stable crystal structures by optimizing the formation energy in the latent space. I strongly believe this research direction has a great potential in material science, especially autonomous materials discovery with desired properties.

7.3.3 *Molecule optimization by deep generative models*

Developing a new drug from original idea to a finished product is a complex, time-consuming and expensive process. In this process, lead optimization plays an important role as enhancing the most promising compounds to improve effectiveness, safety and tolerability (Hughes et al., 2010). I plan to apply graph generative models, for instance our MGVAE (Hy and Kondor, 2021b), into lead optimization. In short, given a prior as the desired property we want to improve in a compound, MGVAE encodes a starting compound or a group of small molecules, and generates a new compound with some structural changes accordingly. If successful, this can replace the computationally expensive molecule synthesis procedure.

7.4 Multiresolution tensor factorization and hierarchical data visualization

The current Pythonic implementation of the learning multiresolution matrix factorization in PyTorch framework (Paszke et al., 2019) at https://github.com/risilab/Learnable_MMF is limited to small and middle size matrices. I plan to implement a large-scale version in C++/CUDA that is both time and memory efficient. I also want to extend this framework into learning to factorize tensors in a multiresolution manner by manifold optimization. Furthermore, besides the wavelet networks learning graphs, another potential application of MMF’s sparse wavelet transform is data visualization. I want to propose a “wavelet” version of t-SNE (van der Maaten and Hinton, 2008) in which instead of optimizing directly

the low-dimensional maps as in the original t-SNE, we will optimize a simple parameterized wavelet network built based on the wavelet basis from MMF factorization of the similarity matrix. The wavelet networks is a promising direction to understand and visualize complex hierarchical structures such as social networks and biological data.

CHAPTER 8

THESIS CONCLUSION

The objective of this thesis is to deepen our understanding of symmetry-preserving, group equivariant, and multiscale/multiresolution neural networks for the purpose of modeling, learning and generating graphs, hypergraphs, N-Body systems, hierarchical structures and highly symmetric structures. This thesis includes a summary of the field of graph learning including several forms of graph kernels and graph neural networks. We have discussed the importance of permutation equivariance and our general architecture, *Covariant Compositional Networks*, that can be efficiently implemented by our new deep learning framework, GraphFlow. Based on the higher order message passing, hierarchical variational autoencoder and a new learning to cluster algorithm, this thesis proposes *Multiresolution Equivariant Graph Variational Autoencoder*, the first ever generative model to generate graphs in both multiresolution and equivariant manner. In addition to permutation equivariance, we also discuss our *Covariant Molecular Neural Networks*, a rotationally equivariant architecture for learning the behavior and properties of complex many-body physical systems. Furthermore, this thesis introduces a new learning algorithm to solve *Multiresolution Matrix Factorization* by a combination of Reinforcement Learning and Stiefel manifold optimization, that results to a new way of getting sparse graph wavelets to construct *Wavelet Neural Networks* learning graphs on spectral domain. Several experiments include molecular properties prediction, learning to estimate the computationally expensive Density Functional Theory calculation, graph & node classification, molecular/general graph generation, citation link prediction, semi-supervised learning molecular representation, graph-based image generation, etc. The source codes and datasets have been made publicly available. This thesis enables various future interdisciplinary research directions of bringing machine learning and artificial intelligence into advancing frontiers of drug discovery, computational biology, material science and theoretical physics.

REFERENCES

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., and Y. Yu, X. Z. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. <https://arxiv.org/abs/1603.04467>.
- Aguiar, J. A., Gong, M. L., Unocic, R. R., Tasdizen, T., and Miller, B. D. (2019). Decoding crystallography from high-resolution electron imaging and diffraction datasets with deep learning. *Science Advances*, 5(10):eaaw1949.
- Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., Bengio, Y., Bergeron, A., Bergstra, J., Bisson, V., Snyder, J. B., Bouchard, N., Boulanger-Lewandowski, N., Bouthillier, X., de Brébisson, A., Breuleux, O., Carrier, P., Cho, K., Chorowski, J., Christiano, P., Cooijmans, T., Cote, M., Cote, M., Courville, A., Dauphin, Y. N., Delalleau, O., Demouth, J., Desjardins, G., Dieleman, S., Dinh, L., Ducoffe, M., Dumoulin, V., Kahou, S. E., Erhan, D., Fan, Z., Firat, O., Germain, M., and Glorot, X. (2016). Theano: A python framework for fast computation of mathematical expressions. <https://arxiv.org/abs/1605.02688>.
- Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97.
- Alley, E., Khimulya, G., Biswas, S., Alquraishi, M., and Church, G. (2019). Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods*, 16.
- Anderson, B., Hy, T. S., and Kondor, R. (2019). Cormorant: Covariant molecular neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Atwood, J. and Towsley, D. (2016). Diffusion-convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 2001–2009, Red Hook, NY, USA. Curran Associates Inc.
- Babai, L. and Kucera, L. (1979). Canonical labelling of graphs in linear average time. *Proceedings Symposium on Foundations of Computer Science*, pages 39–46.
- Bartók, A. P., De, S., Poelking, C., Bernstein, N., Kermode, J. R., Csányi, G., and Ceriotti, M. (2017). Machine learning unifies the modeling of materials and molecules. *Science Advances*, 3(12).

- Bartók, A. P., Kondor, R., and Csányi, G. (2013). On representing chemical environments. *Phys. Rev. B*, 87.
- Bartók, A. P., Payne, M. C., Kondor, R., and Csányi, G. (2010). Gaussian Approximation Potentials: the accuracy of quantum mechanics, without the electrons. *Phys Rev Lett*, 104(13):136403.
- Battaglia, P., Pascanu, R., Lai, M., Jimenez Rezende, D., and kavukcuoglu, k. (2016). Interaction networks for learning about objects, relations and physics. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.
- Behler, J. (2011). Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *J. Chem. Phys.*, 134(7):074106.
- Behler, J. and Parrinello, M. (2007). Generalized neural-network representation of high-dimensional potential-energy surfaces. *Phys. Rev. Lett.*, 98:146401.
- Belkin, M., Niyogi, P., and Sindhwan, V. (2006). Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7(85):2399–2434.
- Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940.
- Bepler, T. and Berger, B. (2019). Learning protein sequence embeddings using information from structure. *CoRR*, abs/1902.08661.
- Beylkin, G., Coifman, R. R., and Rokhlin, V. (1991). Fast wavelet transforms and numerical algorithms i. *Communications on Pure and Applied Mathematics*, 44:141–183.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.
- Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022.
- Bogatskiy, A., Anderson, B., Offermann, J., Roussi, M., Miller, D., and Kondor, R. (2020). Lorentz group equivariant neural network for particle physics. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 992–1002. PMLR.
- Borgwardt, K. M. and Kriegel, H. P. (2005). Shortest-path kernels on graphs. 5:74–81.
- Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A., and Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21 Suppl 1:i47–56.

- Bottou, L., Chapelle, O., DeCoste, D., and Weston, J. (2007). *Support Vector Machine Solvers*, pages 1–27.
- Braun, N. (2019). *Combinatorial Kalman Filter*, pages 117–174. Springer International Publishing, Cham.
- Briandais, R. D. L. (1959). File searching using variable length keys. *Proceedings of the Western Joint Computer Conference*, pages 295–298.
- Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42.
- Brooks, B. R., Brooks, C. L., Mackerell, A. D., Nilsson, L., Petrella, R. J., Roux, B., Won, Y., Archontis, G., Bartels, C., Boresch, S., and et al. (2009). CHARMM: the biomolecular simulation program. *Journal of Computational Chemistry*, 30(10):1545–1614.
- Brooks, B. R., Bruccoleri, R. E., Olafson, B. D., States, D. J., Swaminathan, S., and Karplus, M. (1983). CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4(2):187–217.
- Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. (2014). Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*.
- Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167.
- Cai, J. Y., Furer, M., and Immerman, N. (1992). An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410.
- Cao, N. D. and Kipf, T. (2018). Molgan: An implicit generative model for small molecular graphs.
- Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2016). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *Neural Information Processing Systems (NIPS) Workshop*.
- Chen, X., Cheng, X., and Mallat, S. (2014). Unsupervised deep haar scattering on graphs. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.
- Cheng, X., Chen, X., and Mallat, S. (2015). Deep haar scattering networks. *CoRR*, abs/1509.09187.
- Chiang, K.-Y., Whang, J. J., and Dhillon, I. S. (2012). Scalable clustering of signed networks using balance normalized cut. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM ’12, page 615–624, New York, NY, USA. Association for Computing Machinery.

Chmiela, S., Tkatchenko, A., Sauceda, H. E., Poltavsky, I., Schütt, K. T., and Müller, K.-R. (2016). Machine Learning of Accurate Energy-Conserving Molecular Force Fields. (May):1–6.

Clausen, M. and Baum, U. (1993). Fast fourier transforms for symmetric groups : theory and implementation. *Mathematics of Computation*, 61:833–847.

Cohen, T. and Welling, M. (2017). Steerable CNNs. *Proc. ICLR*, 5.

Cohen, T. S., Geiger, M., Köhler, J., and Welling, M. (2018). Spherical CNNs. *International Conference on Learning Representations*.

Cohen, T. S., Weiler, M., Kicanaoglu, B., and Welling, M. (2019). Gauge equivariant convolutional networks and the icosahedral CNN. *CoRR*, abs/1902.04615.

Cohen, T. S. and Welling, M. (2016). Group equivariant convolutional networks. *Proceedings of The 33rd International Conference on Machine Learning*, 48:2990–2999.

Coifman, R. R. and Maggioni, M. (2006). Diffusion wavelets. *Applied and Computational Harmonic Analysis*, 21(1):53–94. Special Issue: Diffusion Maps and Wavelets.

Collaboration, T. C., Chatrchyan, S., Hmayakyan, G., Khachatryan, V., Sirunyan, A. M., Adam, W., Bauer, T., Bergauer, T., Bergauer, H., Dragicevic, M., Erö, J., Friedl, M., Frühwirth, R., Ghete, V. M., Glaser, P., Hartl, C., Hoermann, N., Hrubec, J., Hänsel, S., Jeitler, M., Kastner, K., Krammer, M., de Abril, I. M., Markytan, M., Mikulec, I., Neuherz, B., Nöbauer, T., Oberegger, M., Padrtá, M., Pernicka, M., Porth, P., Rohringer, H., Schmid, S., Schreiner, T., Stark, R., Steininger, H., Strauss, J., Taurok, A., Uhl, D., Waltenberger, W., Walzel, G., Widl, E., Wulz, C.-E., Petrov, V., Prosolovich, V., Chekhovsky, V., Dvornikov, O., Emelianchik, I., Litomin, A., Makarenko, V., Marfin, I., Mossolov, V., Shumeiko, N., Solin, A., Stefanovitch, R., Gonzalez, J. S., Tikhonov, A., Fedorov, A., Korzhik, M., Mishevitch, O., Zuyeuski, R., Beaumont, W., Cardaci, M., Langhe, E. D., Wolf, E. A. D., Delmeire, E., Ochesanu, S., Tasevsky, M., Mechelen, P. V., D'Hondt, J., Weirdt, S. D., Devroede, O., Goorens, R., Hannaert, S., Heyninck, J., Maes, J., Mozer, M. U., Tavernier, S., Doninck, W. V., Lancker, L. V., Mulders, P. V., Villella, I., Wastiels, C., Yu, C., Bouhali, O., Charaf, O., Clerbaux, B., Harenne, P. D., Lentdecker, G. D., Dewulf, J. P., Elgammal, S., Gindroz, R., Hammad, G. H., Mahmoud, T., Neukermans, L., Pins, M., Pins, R., Rugovac, S., Stefanescu, J., Sundararajan, V., Velde, C. V., Vanlaer, P., Wickens, J., Tytgat, M., Assouak, S., Bonnet, J. L., Bruno, G., Caudron, J., Callatay, B. D., Jeneret, J. D. F. D., Visscher, S. D., Demin, P., Favart, D., Felix, C., Florins, B., Forton, E., Giannanco, A., Grégoire, G., Jonckman, M., Kcira, D., Keutgen, T., Lemaitre, V., Michotte, D., Militaru, O., Ovyn, S., Pierzchala, T., Piotrzkowski, K., Roberfroid, V., Rouby, X., Schul, N., der Aa, O. V., Beliy, N., Daubie, E., Herquet, P., Alves, G., Pol, M. E., Souza, M. H. G., Vaz, M., Damiao, D. D. J., Oguri, V., Santoro, A., Sznajder, A., Gregores, E. D. M., Iope, R. L., Novaes, S. F., Tomei, T., Anguelov, T., Antchev, G., Atanasov, I., Damgov, J., Darmenov, N., Dimitrov, L., Genchev, V., Iaydjiev, P., Marinov, A., Piperov, S., Stoykova, S., Sultanov, G., Trayanov,

R., Vankov, I., Cheshkov, C., Dimitrov, A., Dyulendarova, M., Glushkov, I., Kozhuharov, V., Litov, L., Makariev, M., Marinova, E., Markov, S., Mateev, M., Nasteva, I., Pavlov, B., Petev, P., Petkov, P., Spassov, V., Toteva, Z., Velev, V., Verguilov, V., Bian, J. G., Chen, G. M., Chen, H. S., Chen, M., Jiang, C. H., Liu, B., Shen, X. Y., Sun, H. S., Tao, J., Wang, J., Yang, M., Zhang, Z., Zhao, W. R., Zhuang, H. L., Ban, Y., Cai, J., Ge, Y. C., Liu, S., Liu, H. T., Liu, L., Qian, S. J., Wang, Q., Xue, Z. H., Yang, Z. C., Ye, Y. L., Ying, J., Li, P. J., Liao, J., Xue, Z. L., Yan, D. S., Yuan, H., Montoya, C. A. C., Sanabria, J. C., Godinovic, N., Puljak, I., Soric, I., Antunovic, Z., Dzelalija, M., Marasovic, K., Brigljevic, V., Kadija, K., Morovic, S., Fereos, R., Nicolaou, C., Papadakis, A., Ptochos, F., Razis, P. A., Tsiaakkouri, D., Zinonos, Z., Hektor, A., Kadastik, M., Kannike, K., Lippmaa, E., Müntel, M., Raidal, M., Rebane, L., Aarnio, P. A., Anttila, E., Banzuzi, K., Bulteau, P., Czellar, S., Eiden, N., Eklund, C., Engstrom, P., Heikkinen, A., Honkanen, A., Häkkinen, J., Karimäki, V., Katajisto, H. M., Kinnunen, R., Klem, J., Kortesmaa, J., Kotamäki, M., Kuronen, A., Lampén, T., Lassila-Perini, K., Lefébure, V., Lehti, S., Lindén, T., Luukka, P. R., Michal, S., Brigido, F. M., Mäenpää, T., Nyman, T., Nystén, J., Pietarinen, E., Skog, K., Tammi, K., Tuominen, E., Tuominiemi, J., Ungaro, D., Vanhala, T. P., Wendland, L., Williams, C., Iskanius, M., Korpela, A., Polese, G., Tuuva, T., Bassompierre, G., Bazan, A., David, P. Y., Ditta, J., Drobychev, G., Fouque, N., Guillaud, J. P., Hermel, V., Karneyeu, A., Flour, T. L., Lieunard, S., Maire, M., Mendiburu, P., Nedelec, P., Peigneux, J. P., Schneegans, M., Sillou, D., Vialle, J. P., Anfreville, M., Bard, J. P., Besson, P., Bougamont, E., Boyer, M., Bredy, P., Chipaux, R., Dejardin, M., Denegri, D., Descamps, J., Fabbro, B., Faure, J. L., Ganjour, S., Gentit, F. X., Givernaud, A., Gras, P., de Monchenault, G. H., Jarry, P., Jeanney, C., Kircher, F., Lemaire, M. C., Lemoigne, Y., Levesy, B., Locci, E., Lottin, J. P., Mandjavidze, I., Mur, M., Pansart, J. P., Payn, A., Rander, J., Reymond, J. M., Rolquin, J., Rondeaux, F., Rosowsky, A., Rousse, J. Y. A., Sun, Z. H., Tartas, J., Lysebetten, A. V., Venault, P., Verrecchia, P., Anduze, M., Badier, J., Baffioni, S., Bercher, M., Bernet, C., Berthon, U., Bourotte, J., Busata, A., Busson, P., Cerutti, M., Chamont, D., Charlot, C., Collard, C., Debraine, A., Decotigny, D., Dobrzynski, L., Ferreira, O., Geerebaert, Y., Gilly, J., Gregory, C., Riveros, L. G., Haguenuuer, M., Karar, A., Koblitz, B., Lecouturier, D., Mathieu, A., Milleret, G., Miné, P., Paganini, P., Poilleux, P., Pukhaeva, N., Regnault, N., Romanteau, T., Semeniouk, I., Sirois, Y., Thiebaut, C., Vanel, J. C., Zabi, A., Agram, J. L., Albert, A., Anckenmann, L., Andrea, J., Anstotz, F., Bergdolt, A. M., Berst, J. D., Blaes, R., Bloch, D., Brom, J. M., Cailleret, J., Charles, F., Christophe, E., Claus, G., Coffin, J., Colledani, C., Croix, J., Dangelser, E., Dick, N., Didierjean, F., Drouhin, F., Dulinski, W., Ernenwein, J. P., Fang, R., Fontaine, J. C., Gaudiot, G., Geist, W., Gelé, D., Goeltzenlichter, T., Goerlach, U., Graehling, P., Gross, L., Hu, C. G., Helleboid, J. M., Henkes, T., Hoffer, M., Hoffmann, C., Hosselet, J., Houchu, L., Hu, Y., Huss, D., Illinger, C., Jeanneau, F., Juillot, P., Kachelhoffer, T., Kapp, M. R., Kettunen, H., Ayat, L. L., Bihan, A. C. L., Lounis, A., Maazouzi, C., Mack, V., Majewski, P., Mangeol, D., Michel, J., Moreau, S., Olivetto, C., Pallarès, A., Patois, Y., Pralavorio, P., Racca, C., Riahi, Y., Ripp-Baudot, I., Schmitt, P., Schunck, J. P., Schuster, G., Schwaller, B., Sigward, M. H., Sohler, J. L., Speck, J., Strub, R., Todorov, T., Turchetta, R., Hove, P. V., Vintache, D.,

Zghiche, A., Ageron, M., Augustin, J. E., Baty, C., Baulieu, G., Bedjidian, M., Blaha, J., Bonnevaux, A., Boudoul, G., Brunet, P., Chabanat, E., Chabert, E. C., Chierici, R., Chorowicz, V., Combaret, C., Contardo, D., Negra, R. D., Depasse, P., Drapier, O., Dupanloup, M., Dupasquier, T., Mamouni, H. E., Estre, N., Fay, J., Gascon, S., Giraud, N., Girerd, C., Guillot, G., Haroutunian, R., Ille, B., Lethuillier, M., Lumb, N., Martin, C., Mathez, H., Maurelli, G., Muanza, S., Pangaud, P., Perries, S., Ravat, O., Schibler, E., Schirra, F., Smadja, G., Tissot, S., Trocme, B., Vanzetto, S., Walder, J. P., Bagaturia, Y., Mjavia, D., Mzhavia, A., Tsamalaidze, Z., Roinishvili, V., Adolphi, R., Anagnostou, G., Brauer, R., Braunschweig, W., Esser, H., Feld, L., Karpinski, W., Khomich, A., Klein, K., Kukulies, C., Lübelsmeyer, K., Olzem, J., Ostaptchouk, A., Pandoulas, D., Pierschel, G., Raupach, F., Schael, S., von Dratzig, A. S., Schwering, G., Siedling, R., Thomas, M., Weber, M., Wittmer, B., Wlochal, M., Adamczyk, F., Adolf, A., Altenhöfer, G., Bechstein, S., Bethke, S., Biallass, P., Biebel, O., Bontenackels, M., Bosseler, K., Böhm, A., Erdmann, M., Faissner, H., Fehr, B., Fesefeldt, H., Fetschenhauer, G., Frangenheim, J., Frohn, J. H., Grooten, J., Hebbeker, T., Hermann, S., Hermens, E., Hilgers, G., Hoepfner, K., Hof, C., Jacobi, E., Kappler, S., Kirsch, M., Kreuzer, P., Kupper, R., Lampe, H. R., Lanske, D., Mameghani, R., Meyer, A., Meyer, S., Moers, T., Müller, E., Pahlke, R., Philipps, B., Rein, D., Reithler, H., Reuter, W., Rütten, P., Schulz, S., Schwarthoff, H., Sobek, W., Sowa, M., Stapelberg, T., Szczesny, H., Teykal, H., Teyssier, D., Tomme, H., Tomme, W., Tonutti, M., Tsigenov, O., Tutas, J., Vandenhirtz, J., Wagner, H., Wegner, M., Zeidler, C., Beissel, F., Davids, M., Duda, M., Flügge, G., Giffels, M., Hermanns, T., Heydhausen, D., Kalinin, S., Kasselmann, S., Kaussen, G., Kress, T., Linn, A., Nowack, A., Perchalla, L., Poettgens, M., Pooth, O., Sauerland, P., Stahl, A., Tornier, D., Zoeller, M. H., Behrens, U., Borras, K., Flossdorf, A., Hatton, D., Hegner, B., Kasemann, M., Mankel, R., Meyer, A., Mnich, J., Rosemann, C., Youngman, C., Zeuner, W. D., Bechtel, F., Buhmann, P., Butz, E., Flucke, G., Hamdorf, R. H., Holm, U., Klanner, R., Pein, U., Schirm, N., Schleper, P., Steinbrück, G., Staa, R. V., Wolf, R., Atz, B., Barvich, T., Blüm, P., Boegelspacher, F., Bol, H., Chen, Z. Y., Chowdhury, S., Boer, W. D., Dehm, P., Dirkes, G., Fahrer, M., Felzmann, U., Frey, M., Furgeri, A., Gregoriev, E., Hartmann, F., Hauler, F., Heier, S., Kärcher, K., Ledermann, B., Mueller, S., Müller, T., Neuberger, D., Piasecki, C., Quast, G., Rabbertz, K., Sabellek, A., Scheurer, A., Schilling, F. P., Simonis, H. J., Skiba, A., Steck, P., Theel, A., Thümmel, W. H., Trunov, A., Vest, A., Weiler, T., Weiser, C., Weseler, S., Zhukov, V., Barone, M., Daskalakis, G., Dimitriou, N., Fanourakis, G., Filippidis, C., Geralis, T., Kalfas, C., Karafasoulis, K., Koimas, A., Kyriakis, A., Kyriazopoulou, S., Loukas, D., Markou, A., Markou, C., Mastroyiannopoulos, N., Mavrommatis, C., Mousa, J., Papadakis, I., Petrakou, E., Siotis, I., Theofilatos, K., Tzamarias, S., Vayaki, A., Vermisoglou, G., Zachariadou, A., Gouskos, L., Karapostoli, G., Katsas, P., Panagiotou, A., Papadimitropoulos, C., Aslanoglou, X., Evangelou, I., Kokkas, P., Manthos, N., Papadopoulos, I., Triantis, F. A., Bencze, G., Boldizsar, L., Debreczeni, G., Hajdu, C., Hidas, P., Horvath, D., Kovacsarki, P., Laszlo, A., Odor, G., Patay, G., Sikler, F., Veres, G., Veszterombi, G., Zalan, P., Fenyvesi, A., Imrek, J., Molnar, J., Novak, D., Palinkas, J., Szekely, G., Beni, N., Kapusi, A., Marian, G., Radics, B., Raics, P., Szabo, Z., Szillasi, Z., Trocsanyi, Z. L., Zilizi, G., Bawa, H. S.,

Beri, S. B., Bhandari, V., Bhatnagar, V., Kaur, M., Kohli, J. M., Kumar, A., Singh, B., Singh, J. B., Arora, S., Bhattacharya, S., Chatterji, S., Chauhan, S., Choudhary, B. C., Gupta, P., Jha, M., Ranjan, K., Shivpuri, R. K., Srivastava, A. K., Choudhury, R. K., Dutta, D., Ghodgaonkar, M., Kailas, S., Kataria, S. K., Mohanty, A. K., Pant, L. M., Shukla, P., Topkar, A., Aziz, T., Banerjee, S., Bose, S., Chendvankar, S., Deshpande, P. V., Guchait, M., Gurtu, A., Maity, M., Majumder, G., Mazumdar, K., Nayak, A., Patil, M. R., Sharma, S., Sudhakar, K., Acharya, B. S., Banerjee, S., Bheesette, S., Dugad, S., Kalmani, S. D., Lakkireddi, V. R., Mondal, N. K., Panyam, N., Verma, P., Arfaei, H., Hashemi, M., Najafabadi, M. M., Moshaii, A., Mehdiabadi, S. P., Felcini, M., Grunewald, M., Abadjiev, K., Abbrescia, M., Barbone, L., Cariola, P., Chiumarulo, F., Clemente, A., Colaleo, A., Creanza, D., Filippis, N. D., Palma, M. D., Robertis, G. D., Donvito, G., Ferorelli, R., Fiore, L., Franco, M., Giordano, D., Guida, R., Iaselli, G., Lacalamita, N., Loddo, F., Maggi, G., Maggi, M., Manna, N., Marangelli, B., Mennea, M. S., My, S., Natali, S., Nuzzo, S., Papagni, G., Pinto, C., Pompili, A., Pugliese, G., Ranieri, A., Romano, F., Roselli, G., Sala, G., Selvaggi, G., Silvestris, L., Tempesta, P., Trentadue, R., Tupputi, S., Zito, G., Abbiendi, G., Bacchi, W., Battilana, C., Benvenuti, A. C., Boldini, M., Bonacorsi, D., Braibant-Giacomelli, S., Cafaro, V. D., Capiluppi, P., Castro, A., Cavallo, F. R., Ciocca, C., Codispoti, G., Cuffiani, M., D'Antone, I., Dallavalle, G. M., Fabbri, F., Fanfani, A., Finelli, S., Giacomelli, P., Giordano, V., Giunta, M., Grandi, C., Guerzoni, M., Guiducci, L., Marcellini, S., Masetti, G., Montanari, A., Navarría, F. L., Odorici, F., Paolucci, A., Pellegrini, G., Perrotta, A., Rossi, A. M., Rovelli, T., Siroli, G. P., Torromeo, G., Travaglini, R., Veronese, G. P., Albergo, S., Chiorboli, M., Costa, S., Galanti, M., Rotondo, G. G., Giudice, N., Guardone, N., Noto, F., Potenza, R., Saizu, M. A., Salemi, G., Sutera, C., Tricomi, A., Tuve, C., Bellucci, L., Brianzi, M., Broccolo, G., Catacchini, E., Ciulli, V., Civinini, C., D'Alessandro, R., Focardi, E., Frosali, S., Genta, C., Landi, G., Lenzi, P., Macchiolo, A., Maletta, F., Manolescu, F., Marchettini, C., Masetti, L., Mersi, S., Meschini, M., Minelli, C., Paoletti, S., Parrini, G., Scarlini, E., Sguazzoni, G., Benussi, L., Bertani, M., Bianco, S., Caponero, M., Colonna, D., Daniello, L., Fabbri, F., Felli, F., Giardoni, M., Monaca, A. L., Ortenzi, B., Pallotta, M., Paolozzi, A., Paris, C., Passamonti, L., Pierluigi, D., Ponzio, B., Pucci, C., Russo, A., Saviano, G., Fabbricatore, P., Farinon, S., Greco, M., Musenich, R., Badoer, S., Berti, L., Biasotto, M., Fantinel, S., Frizziero, E., Gastaldi, U., Gulmini, M., Lelli, F., Maron, G., Squizzato, S., Toniolo, N., Traldi, S., Banfi, S., Bertoni, R., Bonesini, M., Carbone, L., Cerati, G. B., Chignoli, F., D'Angelo, P., Min, A. D., Dini, P., Farina, F. M., Ferri, F., Govoni, P., Magni, S., Malberti, M., Malvezzi, S., Mazza, R., Menasce, D., Miccio, V., Moroni, L., Negri, P., Paganoni, M., Pedrini, D., Pullia, A., Ragazzi, S., Redaelli, N., Rovere, M., Sala, L., Sala, S., Salerno, R., de Fatis, T. T., Tancini, V., Taroni, S., Boiano, A., Cassese, F., Cassese, C., Cimmino, A., D'Aquino, B., Lista, L., Lomidze, D., Noli, P., Paolucci, P., Passeggio, G., Piccolo, D., Roscilli, L., Sciacca, C., Vanzanella, A., Azzi, P., Bacchetta, N., Barcellan, L., Bellato, M., Benettoni, M., Bisello, D., Borsato, E., Candelori, A., Carlin, R., Castellani, L., Checchia, P., Ciano, L., Colombo, A., Conti, E., Rold, M. D., Corso, F. D., Giorgi, M. D., Mattia, M. D., Dorigo, T., Dosselli, U., Fanin, C., Galet, G., Gasparini, F., Gasparini, U., Giraldo, A.,

Giubilato, P., Gonella, F., Gresele, A., Griggio, A., Guaita, P., Kaminskiy, A., Karaevskii, S., Khomenkov, V., Kostylev, D., Lacaprara, S., Lazzizza, I., Lippi, I., Loreti, M., Margoni, M., Martinelli, R., Mattiazzo, S., Mazzucato, M., Meneguzzo, A. T., Modenese, L., Montecassiano, F., Neviani, A., Nigro, M., Paccagnella, A., Pantano, D., Parenti, A., Passaseo, M., Pedrotta, R., Pegoraro, M., Rampazzo, G., Reznikov, S., Ronchese, P., Daponte, A. S., Sartori, P., Stavitskiy, I., Tessaro, M., Torassa, E., Triossi, A., Vanini, S., Ventura, S., Ventura, L., Verlato, M., Zago, M., Zatti, F., Zotto, P., Zumerle, G., Baesso, P., Belli, G., Berzano, U., Bricola, S., Grelli, A., Musitelli, G., Nardò, R., Necchi, M. M., Pagano, D., Ratti, S. P., Riccardi, C., Torre, P., Vicini, A., Vitulo, P., Viviani, C., Aisa, D., Aisa, S., Ambroglini, F., Angarano, M. M., Babucci, E., Benedetti, D., Biasini, M., Bilei, G. M., Bizzaglia, S., Brunetti, M. T., Caponeri, B., Checcucci, B., Covarelli, R., Dinu, N., Fanò, L., Farnesini, L., Giorgi, M., Lariccia, P., Mantovani, G., Moscatelli, F., Passeri, D., Piluso, A., Placidi, P., Postolache, V., Santinelli, R., Santocchia, A., Servoli, L., Spiga, D., Azzurri, P., Bagliesi, G., Balestri, G., Basti, A., Bellazzini, R., Benucci, L., Bernardini, J., Berretta, L., Bianucci, S., Boccali, T., Bocci, A., Borrello, L., Bosi, F., Bracci, F., Brez, A., Calzolari, F., Castaldi, R., Cazzola, U., Ceccanti, M., Cecchi, R., Cerri, C., Cucoanes, A. S., Dell'Orso, R., Dobur, D., Dutta, S., Fiori, F., Foà, L., Gaggelli, A., Gennai, S., Giassi, A., Giusti, S., Kartashov, D., Kraan, A., Latronico, L., Ligabue, F., Linari, S., Lomtadze, T., Lungu, G. A., Magazzu, G., Mammini, P., Mariani, F., Martinelli, G., Massa, M., Messineo, A., Moggi, A., Palla, F., Palmonari, F., Petragnani, G., Petrucciani, G., Profeti, A., Raffaelli, F., Rizzi, D., Sanguinetti, G., Sarkar, S., Segneri, G., Sentenac, D., Serban, A. T., Slav, A., Spagnolo, P., Spandre, G., Tenchini, R., Tolaini, S., Tonelli, G., Venturi, A., Verdini, P. G., Vos, M., Zaccarelli, L., Baccaro, S., Barone, L., Bartoloni, A., Borgia, B., Capradossi, G., Cavallari, F., Cecilia, A., D'Angelo, D., Dafinei, I., Re, D. D., Marco, E. D., Diemoz, M., Ferrara, G., Gargiulo, C., Guerra, S., Iannone, M., Longo, E., Montecchi, M., Nuccetelli, M., Organtini, G., Palma, A., Paramatti, R., Pellegrino, F., Rahatlou, S., Rovelli, C., Tehrani, F. S., Zullo, A., Alampi, G., Amapane, N., Arcidiacono, R., Argiro, S., Arneodo, M., Bellan, R., Benotto, F., Biino, C., Bolognesi, S., Borgia, M. A., Botta, C., Brasolin, A., Cartiglia, N., Castello, R., Cerminara, G., Cirio, R., Cordero, M., Costa, M., Dattola, D., Daudo, F., Dellacasa, G., Demaria, N., Dughera, G., Dumitrache, F., Farano, R., Ferrero, G., Filoni, E., Kostyleva, G., Larsen, H. E., Mariotti, C., Marone, M., Maselli, S., Menichetti, E., Mereu, P., Migliore, E., Mila, G., Monaco, V., Musich, M., Nervo, M., Obertino, M. M., Panero, R., Parussa, A., Pastrone, N., Peroni, C., Petrillo, G., Romero, A., Ruspa, M., Sacchi, R., Scalise, M., Solano, A., Staiano, A., Trapani, P. P., Trocino, D., Vaniev, V., Pereira, A. V., Zampieri, A., Belforte, S., Cossutti, F., Ricca, G. D., Gobbo, B., Kavka, C., Penzo, A., Kim, Y. E., Nam, S. K., Kim, D. H., Kim, G. N., Kim, J. C., Kong, D. J., Ro, S. R., Son, D. C., Park, S. Y., Kim, Y. J., Kim, J. Y., Lim, I. T., Pac, M. Y., Lee, S. J., Jung, S. Y., Rhee, J. T., Ahn, S. H., Hong, B. S., Jeng, Y. K., Kang, M. H., Kim, H. C., Kim, J. H., Kim, T. J., Lee, K. S., Lim, J. K., Moon, D. H., Park, I. C., Park, S. K., Ryu, M. S., Sim, K.-S., Son, K. J., Hong, S. J., Choi, Y. I., Valdez, H. C., Hernandez, A. S., Moreno, S. C., Pineda, A. M., Aerts, A., der Stok, P. V., Weffers, H., Allfrey, P., Gray, R. N. C., Hashimoto, M., Korfcheck, D., Bell, A. J., Rodrigues, N. B., Butler, P. H., Churchwell, S., Knegjens, R., Whitehead, S.,

Williams, J. C., Aftab, Z., Ahmad, U., Ahmed, I., Ahmed, W., Asghar, M. I., Asghar, S., Dad, G., Hafeez, M., Hoorani, H. R., Hussain, I., Hussain, N., Iftikhar, M., Khan, M. S., Mehmood, K., Osman, A., Shahzad, H., Zafar, A. R., Ali, A., Bashir, A., Jan, A. M., Kamal, A., Khan, F., Saeed, M., Tanvir, S., Zafar, M. A., Blocki, J., Cyz, A., Gladysz-Dziadus, E., Mikocki, S., Rybczynski, M., Turnau, J., Włodarczyk, Z., Zychowski, P., Bunkowski, K., Cwiok, M., Czyrkowski, H., Dabrowski, R., Dominik, W., Doroba, K., Kalinowski, A., Kierzkowski, K., Konecki, M., Krolikowski, J., Kudla, I. M., Pietrusinski, M., Pozniak, K., Zabolotny, W., Zych, P., Gokieli, R., Goscilo, L., Górska, M., Nawrocki, K., Traczyk, P., Wrochna, G., Zalewski, P., Pozniak, K. T., Romaniuk, R., Zabolotny, W. M., Alemany-Fernandez, R., Almeida, C., Almeida, N., Verde, A. S. A. V., Monteiro, T. B., Bluj, M., Silva, S. D. M., Mendes, A. D. T., Ferreira, M. F., Gallinaro, M., Husejko, M., Jain, A., Kazana, M., Musella, P., Nobrega, R., Silva, J. R. D., Ribeiro, P. Q., Santos, M., Silva, P., Silva, S., Teixeira, I., Teixeira, J. P., Varela, J., Varner, G., Cardoso, N. V., Altsybeev, I., Babich, K., Belkov, A., Belotelov, I., Bunin, P., Chesnevskaya, S., Elsha, V., Ershov, Y., Filozova, I., Finger, M., Finger, M., Golunov, A., Golutvin, I., Gorbounov, N., Gramenitski, I., Kalagin, V., Kamenev, A., Karjavin, V., Khabarov, S., Khabarov, V., Kiryushin, Y., Konoplyanikov, V., Korenkov, V., Kozlov, G., Kurenkov, A., Lanev, A., Lysiakov, V., Malakhov, A., Melnichenko, I., Mitsyn, V. V., Moisenz, K., Moisenz, P., Movchan, S., Nikonorov, E., Oleynik, D., Palichik, V., Perelygin, V., Petrosyan, A., Rogalev, E., Samsonov, V., Savina, M., Semenov, R., Sergeev, S., Shmatov, S., Shulha, S., Smirnov, V., Smolin, D., Tcheremoukhine, A., Teryaev, O., Tikhonenko, E., Urkinbaev, A., Vasil'ev, S., Vishnevskiy, A., Volodko, A., Zamiatin, N., Zarubin, A., Zarubin, P., Zubarev, E., Bondar, N., Gavrikov, Y., Golovtsov, V., Ivanov, Y., Kim, V., Kozlov, V., Lebedev, V., Makarenkov, G., Moroz, F., Neustroev, P., Obrant, G., Orishchin, E., Petrunin, A., Shcheglov, Y., Shchetkovskiy, A., Sknar, V., Skorobogatov, V., Smirnov, I., Sulimov, V., Tarakanov, V., Uvarov, L., Vavilov, S., Velichko, G., Volkov, S., Vorobyev, A., Chmelev, D., Druzhkin, D., Ivanov, A., Kudinov, V., Logatchev, O., Onishchenko, S., Orlov, A., Sakharov, V., Smetannikov, V., Tikhomirov, A., Zavodthikov, S., Andreev, Y., Anisimov, A., Duk, V., Gninenko, S., Golubev, N., Gorbunov, D., Kirsanov, M., Krasnikov, N., Matveev, V., Pashenkov, A., Pastsyak, A., Postoev, V. E., Sadovski, A., Skassyrskaya, A., Solovey, A., Solovey, A., Soloviev, D., Toropin, A., Troitsky, S., Alekhin, A., Baldov, A., Epshteyn, V., Gavrilov, V., Ilina, N., Kaftanov, V., Karpishin, V., Kiselevich, I., Kolosov, V., Kossov, M., Krokhotin, A., Kuleshov, S., Oulianov, A., Pozdnyakov, A., Safronov, G., Semenov, S., Stepanov, N., Stolin, V., Vlasov, E., Zaytsev, V., Boos, E., Dubinin, M., Dudko, L., Ershov, A., Eyyubova, G., Gribushin, A., Ilyin, V., Klyukhin, V., Kodolova, O., Kruglov, N. A., Kryukov, A., Loktin, I., Malinina, L., Mikhaylin, V., Petrushanko, S., Sarycheva, L., Savrin, V., Shamardin, L., Sherstnev, A., Snigirev, A., Teplov, K., Vardanyan, I., Fomenko, A. M., Konovalova, N., Kozlov, V., Lebedev, A. I., Lvova, N., Rusakov, S. V., Terkulov, A., Abramov, V., Akimenko, S., Artamonov, A., Ashimova, A., Azhgirey, I., Bitioukov, S., Chikilev, O., Datsko, K., Filine, A., Godizov, A., Goncharov, P., Grishin, V., Inyakin, A., Kachanov, V., Kalinin, A., Khmelnikov, A., Konstantinov, D., Korablev, A., Krychkine, V., Krinitzsyn, A., Levine, A., Lobov, I., Lukyanin, V., Mel'nik, Y., Molchanov, V., Petrov, V., Petukhov, V., Pikalov, V., Ryazanov,

A., Ryutin, R., Shelikhov, V., Skvortsov, V., Slabospitsky, S., Sobol, A., Sytine, A., Talov, V., Tourtchanovitch, L., Troshin, S., Tyurin, N., Uzunian, A., Volkov, A., Zelepoukine, S., Lukyanov, V., Mamaeva, G., Prilutskaya, Z., Rumyantsev, I., Sokha, S., Tataurschikov, S., Vasilyev, I., Adzic, P., Anicin, I., Djordjevic, M., Jovanovic, D., Maletic, D., Puzovic, J., Smiljkovic, N., Navarrete, E. A., Aguilar-Benitez, M., Munoz, J. A., Vega, J. M. A., Alberdi, J., Maestre, J. A., Martin, M. A., Arce, P., Barcala, J. M., Berdugo, J., Ramos, C. L. B., Lazaro, C. B., Bejar, J. C., Calvo, E., Cerrada, M., Llatas, M. C., Catalán, J. J. C., Colino, N., Daniel, M., Cruz, B. D. L., Peris, A. D., Bedoya, C. F., Ferrando, A., Fouz, M. C., Ferrero, D. F., Romero, J. G., Garcia-Abia, P., Lopez, O. G., Hernandez, J. M., Josa, M. I., Marin, J., Merino, G., Molinero, A., Navarrete, J. J., Oller, J. C., Pelayo, J. P., Sanchez, J. C. P., Ramirez, J., Romero, L., Munoz, C. V., Willmott, C., Yuste, C., Albajar, C., de Trocóniz, J. F., Jimenez, I., Macias, R., Teixeira, R. F., Cuevas, J., Menéndez, J. F., Caballero, I. G., Lopez-Garcia, J., Sordo, H. N., Garcia, J. M. V., Cabrillo, I. J., Calderon, A., Fernandez, D. C., Merino, I. D., Campderros, J. D., Fernandez, M., Menendez, J. F., Figueroa, C., Moral, L. A. G., Gomez, G., Casademunt, F. G., Sanchez, J. G., Suarez, R. G., Jorda, C., Pardo, P. L., Garcia, A. L., Virtó, A. L., Marco, J., Marco, R., Rivero, C. M., del Arbol, P. M. R., Matorras, F., Fernandez, P. O., Revuelta, A. P., Rodrigo, T., Gonzalez, D. R., Jimeno, A. R., Scodellaro, L., Sanudo, M. S., Vila, I., Cortabitarte, R. V., Barbero, M., Goldin, D., Henrich, B., Tauscher, L., Vlachos, S., Wadhwa, M., Abbaneo, D., Abbas, S. M., Ahmed, I., Akhtar, S., Akhtar, M. I., Albert, E., Alidra, M., Ashby, S., Aspell, P., Auffray, E., Baillon, P., Ball, A., Bally, S. L., Bangert, N., Barillère, R., Barney, D., Beauceron, S., Beaudette, F., Benelli, G., Benetta, R., Benichou, J. L., Bialas, W., Bjorkbo, A., Blechschmidt, D., Bloch, C., Bloch, P., Bonacini, S., Bos, J., Bosteels, M., Boyer, V., Branson, A., Breuker, H., Bruneliere, R., Buchmuller, O., Campi, D., Camporesi, T., Caner, A., Cano, E., Carrone, E., Cattai, A., Chatelain, J. P., Chauvey, M., Christiansen, T., Ciganek, M., Cittolin, S., Cogan, J., Garcia, A. C., Cornet, H., Corrin, E., Corvo, M., Cucciarelli, S., Curé, B., D'Enterria, D., Roeck, A. D., de Visser, T., Delaere, C., Delattre, M., Deldicque, C., Delikaris, D., Deyrail, D., Vincenzo, S. D., Domeniconi, A., Santos, S. D., Duthion, G., Edera, L. M., Elliott-Peisert, A., Eppard, M., Fanzago, F., Favre, M., Foeth, H., Folch, R., Frank, N., Fratianni, S., Freire, M. A., Frey, A., Fucci, A., Funk, W., Gaddi, A., Gagliardi, F., Gastal, M., Gateau, M., Gayde, J. C., Gerwig, H., Ghezzi, A., Gigi, D., Gill, K., Giolo-Nicollerat, A. S., Girod, J. P., Glege, F., Glessing, W., Garrido, R. G.-R., Goudard, R., Grabit, R., Grillet, J. P., Llamas, P. G., Mlot, E. G., Gutleber, J., Hall-wilton, R., Hammarstrom, R., Hansen, M., Harvey, J., Hervé, A., Hill, J., Hoffmann, H. F., Holzner, A., Honma, A., Hufnagel, D., Huhtinen, M., Ilie, S. D., Innocente, V., Jank, W., Janot, P., Jarron, P., Jeanrenaud, M., Jouvel, P., Kerkach, R., Kloukinas, K., Kottelat, L. J., Labbé, J. C., Lacroix, D., Lagrue, X., Lasseur, C., Laure, E., Laurens, J. F., Lazeyras, P., Goff, J. M. L., Lebeau, M., Lecoq, P., Lemeilleur, F., Lenzi, M., Leonardo, N., Leonidopoulos, C., Letheren, M., Liendl, M., Limia-Conde, F., Linssen, L., Ljuslin, C., Lofstedt, B., Loos, R., Perez, J. A. L., Lourenco, C., Lyonnet, A., Machard, A., Mackenzie, R., Magini, N., Maire, G., Malgeri, L., Malina, R., Mannelli, M., Marchioro, A., Martin, J., Meijers, F., Meridiani, P., Meschi, E., Meyer, T., Cordonnier, A. M., Michaud, J. F., Mirabito, L., Moser, R., Mossiere, F., Muffat-

Joly, J., Mulders, M., Mulon, J., Murer, E., Mättig, P., Oh, A., Onnela, A., Oriunno, M., Orsini, L., Osborne, J. A., Paillard, C., Pal, I., Papotti, G., Passardi, G., Patino-Revuelta, A., Patras, V., Solano, B. P., Perez, E., Perinic, G., Pernot, J. F., Petagna, P., Petiot, P., Petit, P., Petrilli, A., Pfeiffer, A., Piccut, C., Pimiä, M., Pintus, R., Pioppi, M., Placci, A., Pollet, L., Postema, H., Price, M. J., Principe, R., Racz, A., Radermacher, E., Ranieri, R., Raymond, G., Rebecchi, P., Rehn, J., Reynaud, S., Naraghi, H. R., Ricci, D., Ridel, M., Risoldi, M., Moreira, P. R. S., Rohlev, A., Roiron, G., Rolandi, G., Rumerio, P., Runolfsson, O., Ryjov, V., Sakulin, H., Samyn, D., Amaral, L. C. S., Sauce, H., Sbrissa, E., Scharff-Hansen, P., Schieferdecker, P., Schlatter, W. D., Schmitt, B., Schmuecker, H. G., Schröder, M., Schwick, C., Schäfer, C., Segoni, I., Roldán, P. S., Sgobba, S., Sharma, A., Siegrist, P., Sigaud, C., Sinanis, N., Sobrier, T., Sphicas, P., Spiropulu, M., Stefanini, G., Strandlie, A., Szoncsó, F., Taylor, B. G., Teller, O., Thea, A., Tournefier, E., Treille, D., Tropea, P., Troska, J., Tsesmelis, E., Tsirou, A., Valls, J., Vulpen, I. V., Donckt, M. V., Vasey, F., Acosta, M. V., Veillet, L., Vichoudis, P., Waurick, G., Wellisch, J. P., Wertelaers, P., Wilhelmsson, M., Willers, I. M., Winkler, M., Zanetti, M., Bertl, W., Deiters, K., Dick, P., Erdmann, W., Feichtinger, D., Gabathuler, K., Hochman, Z., Horisberger, R., Ingram, Q., Kaestli, H. C., Kotlinski, D., König, S., Poerschke, P., Renker, D., Rohe, T., Sakhelashvili, T., Starodumov, A., Aleksandrov, V., Behner, F., Beniozef, I., Betev, B., Blau, B., Brett, A. M., Caminada, L., Chen, Z., Chivarov, N., Calafiori, D. D. S. D., Dambach, S., Davatz, G., Delachenal, V., Marina, R. D., Dimov, H., Dissertori, G., Dittmar, M., Djambazov, L., Dröge, M., Eggel, C., Ehlers, J., Eichler, R., Elmiger, M., Faber, G., Freudenreich, K., Fuchs, J. F., Georgiev, G. M., Grab, C., Haller, C., Herrmann, J., Hilgers, M., Hintz, W., Hofer, H., Hofer, H., Horisberger, U., Horvath, I., Hristov, A., Humbertclaude, C., Iliev, B., Kastli, W., Kruse, A., Kuipers, J., Langenegger, U., Lecomte, P., Lejeune, E., Leshev, G., Lesmond, C., List, B., Luckey, P. D., Lustermann, W., Maillefaud, J. D., Marchica, C., Maurisset, A., Meier, B., Milenovic, P., Milesi, M., Moortgat, F., Nanov, I., Nardulli, A., Nessi-Tedaldi, F., Panev, B., Pape, L., Pauss, F., Petrov, E., Petrov, G., Peynеков, M. M., Pitzl, D., Punz, T., and Riboni, P. (2008). The CMS experiment at the CERN LHC. *Journal of Instrumentation*, 3(08):S08004–S08004.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301.

Dai, H., Nazi, A., Li, Y., Dai, B., and Schuurmans, D. (2020). Scalable deep generative modeling for sparse graphs. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2302–2312. PMLR.

Daubechies, I. (1988). Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 41(7):909–996.

Daubechies, I. (1992). *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, USA.

- Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797.
- Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.
- Dhillon, I., Guan, Y., and Kulis, B. (2005). A fast kernel-based multilevel algorithm for graph clustering. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, page 629–634, New York, NY, USA. Association for Computing Machinery.
- Dhillon, I. S., Guan, Y., and Kulis, B. (2007). Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957.
- Diaconis, P. and Rockmore, D. N. (1990). Efficient computation of the fourier transform on finite groups. *Journal of the American Mathematical Society*, 3:297–332.
- Dieng, A. B., Ruiz, F. J. R., Blei, D. M., and Titsias, M. K. (2019). Prescribed generative adversarial networks.
- Ding, Y., Kondor, R., and Eskreis-Winkler, J. (2017). Multiresolution kernel approximation for gaussian process regression. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Drineas, P., Kannan, R., and Mahoney, M. W. (2006). Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM J. Comput.*, 36:158–183.
- Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. 28:2224–2232.
- Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks. *CoRR*, abs/2003.00982.
- Edmonds, J. and Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*.
- Erdos, P. and Renyi, A. (1960). On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.*, 5:17–61.

- Esteves, C., Allen-Blanchette, C., Makadia, A., and Daniilidis, K. (2017). Learning SO(3) Equivariant Representations with Spherical CNNs.
- Evans, L. and Bryant, P. (2008). LHC machine. *Journal of Instrumentation*, 3(08):S08001–S08001.
- Faber, F. A., Hutchison, L., , Huang, B., Gilmer, J., Schoenholz, S. S., Dahl, G. E., Vinyals, O., Kearnes, S., Riley, P. F., and von Lilienfeld, O. A. (2017). Prediction errors of molecular machine learning models lower than hybrid dft error. *J. Chem. Theory Comput.*, 13:5255 – 5264.
- Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32:541–551.
- Felzenszwalb, P. F. and Huttenlocher, D. P. (2005). Pictorial structures for object recognition. *Int. J. Comput. Vis.*, 61:55–71.
- Feragen, A., Kasenburg, N., Peterson, J., de Brujinne, M., and Borgwardt, K. M. (2013). Scalable kernels for graphs with continuous attributes. 26.
- Ferré, G., Maillet, J.-B., and Stoltz, G. (2015). Permutation-invariant distance between atomic configurations. *J. Chem. Phys.*, 143(10):104114.
- Fischler, M. and Elschlager, R. (1973). The representation and matching of pictorial structures. *IEEE Trans. Comput.*, C-22:67–92.
- Fletcher, R. (1988). Practical methods of optimization.
- Fout, A., Byrd, J., Shariat, B., and Ben-Hur, A. (2017). Protein interface prediction using graph convolutional networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, page 6533–6542, Red Hook, NY, USA. Curran Associates Inc.
- Fredkin, E. (1960). Trie memory. *Communications of the ACM*, 3:490–499.
- Freeman, W. T. and Adelson, E. H. (1991). The design and use of steerable filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13:891–906.
- Gao, W., Mahajan, S. P., Sulam, J., and Gray, J. J. (2020). Deep learning in protein structural modeling and design. *Patterns*, 1.
- Gärtner, T. (2002). Exponential and geometric kernels for graphs.
- Gärtner, T., Flach, P. A., and Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. *Proceedings of the Annual Conference on Computational Learning Theory*, pages 129–143.

- Gavish, M., Nadler, B., and Coifman, R. R. (2010). Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML’10, page 367–374, Madison, WI, USA. Omnipress.
- Getoor, L. (2005). *Link-based Classification*, pages 189–207. Springer London, London.
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR.
- Gittens, A. and Mahoney, M. (2013). Revisiting the nystrom method for improved large-scale machine learning. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 567–575, Atlanta, Georgia, USA. PMLR.
- Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.
- Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276. PMID: 29532027.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.
- Greengard, L. and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348.
- Gumbel, E. J. (1954). Statistical theory of extreme values and some practical applications: a series of lectures. *US Govt. Print. Office*, Number 33.
- Haar, A. (1910). Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69:331–371.
- Hachmann, J., Olivares-Amaya, R., Atahan-Evrenk, S., Amador-Bedolla, C., Sánchez-Carrera, R. S., Gold-Parker, A., Vogt, L., Brockway, A. M., and Aspuru-Guzik, A. (2011). The harvard clean energy project: Large-scale computational screening and design of organic photovoltaics on the world community grid. *The Journal of Physical Chemistry Letters*, 2(17):2241–2251.

- Halko, N., Martinsson, P. G., and Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288.
- Hamilton, W. L. (2020). Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.
- Hammond, D. K., Vandergheynst, P., and Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150.
- Hansen, K., Biegler, F., Ramakrishnan, R., Pronobis, W., von Lilienfeld, O. A., Müller, K.-R., and Tkatchenko, A. (2015). Machine learning predictions of molecular properties: Accurate many-body potentials and non-locality in chemical space. *J. Chem. Phys.*, 6(12):2326–2331.
- Hansen, K., Montavon, G., Biegler, F., Fazli, S., Rupp, M., Scheffler, M., von Lilienfeld, O. A., Tkatchenko, A., and Müller, K.-R. (2013). Assessment and validation of machine learning methods for predicting molecular atomization energies. *J. Chem. Theory Comput.*, 9(8):3404–3419.
- Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.
- Heinzinger, M., Elnaggar, A., Wang, Y., Dallago, C., Nachaev, D., Matthes, F., and Rost, B. (2019). Modeling the language of life – deep learning protein sequences. *bioRxiv*.
- Henaff, M., Bruna, J., and LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.
- Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Hirn, M., Mallat, S., and Poilvert, N. (2017). Wavelet scattering regression of quantum chemical energies. *Multiscale Modeling & Simulation*, 15:827–863.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1735–1780.
- Hofmann, T., Schölkopf, B., and Smola, A. J. (2008). Kernel methods in machine learning. *The Annals of Statistics*, 36:1171–1220.
- Hohenberg, P. and Kohn, W. (1964). Inhomogeneous electron gas. *Phys. Rev.*, 136:864–871.
- Horak, D. and Jost, J. (2013). Spectra of combinatorial laplace operators on simplicial complexes. *Advances in Mathematics*, 244:303–336.

- Huang, B. and von Lilienfeld, O. A. (2016). Communication: Understanding molecular representations in machine learning: The role of uniqueness and target similarity. *J. Chem. Phys.*, 145.
- Hughes, J., Rees, S., Kalindjian, B., and Philpott, K. (2010). Principles of early drug discovery. *British journal of pharmacology*, 162:1239–49.
- Hy, T. S. (2017). Graphflow. *GitHub*.
- Hy, T. S. and Jones, C. (2019). Graph neural networks with efficient tensor operations in cuda/gpu and graphflow deep learning framework in c++ for quantum chemistry.
- Hy, T. S. and Kondor, R. (2021a). Learning multiresolution matrix factorization and its wavelet networks on graphs.
- Hy, T. S. and Kondor, R. (2021b). Multiresolution graph variational autoencoder. *CoRR*, abs/2106. 00967.
- Hy, T. S., Trivedi, S., Pan, H., Anderson, B. M., , and Kondor, R. (2018). Predicting molecular properties with covariant compositional networks. *The Journal of Chemical Physics*, 148.
- Ingraham, J., Garg, V., Barzilay, R., and Jaakkola, T. (2019). Generative models for graph-based protein design. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Ingraham, J. and Marks, D. (2017). Variational inference for sparse and undirected models. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1607–1616. PMLR.
- Ithapu, V. K., Kondor, R., Johnson, S. C., and Singh, V. (2017). The incremental multiresolution matrix factorization algorithm. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 692–701.
- Jacobi, C. (1846). Über ein leichtes verfahren die in der theorie der säcularstörungen vorkommenden gleichungen numerisch aufzulösen*):. 1846(30):51–94.
- Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with gumbel-softmax. In *ICLR*.
- Jebara, T. and Kondor, R. (2003). Bhattacharyya expected likelihood kernels. volume 2777, pages 57–71.
- Jenatton, R., Obozinski, G., and Bach, F. (2010). Structured sparse principal component analysis. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 366–373, Chia Laguna Resort, Sardinia, Italy. PMLR.

- Jin, W., Barzilay, R., and Jaakkola, T. (2018). Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2323–2332. PMLR.
- Jin, W., Coley, C. W., Barzilay, R., and Jaakkola, T. (2017). Predicting organic reaction outcomes with weisfeiler-lehman network. *31st Conference on Neural Information Processing Systems (NIPS)*.
- Joachims, T. (2006). Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’06, page 217–226, New York, NY, USA. Association for Computing Machinery.
- Johansson, F. D. and Dubhashi, D. (2015). Learning with similarity functions on graphs using matchings of geometric embeddings. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’15, page 467–476, New York, NY, USA. Association for Computing Machinery.
- Jonsson, J. (2008). *Simplicial Complexes of Graphs*, volume 1928.
- Kashima, H., Tsuda, K., and Inokuchi, A. (2004). Kernels for graphs. *Kernels and Bioinformatics*, pages 155–170.
- Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608.
- Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proc. ICLR*, San Diego.
- Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.
- Kipf, T. N. and Welling, M. (2016). Variational graph auto-encoders.
- Kipf, T. N. and Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations, ICLR ’17*.
- Kirklin, S., Saal, J. E., Meredig, B., Thompson, A., Doak, J. W., Aykol, M., Rühl, S., and Wolverton, C. (2015). The open quantum materials database (oqmd): assessing the accuracy of dft formation energies. *NPJ Comput. Mat.*, 1:15010.

- Klushyn, A., Chen, N., Kurle, R., Cseke, B., and van der Smagt, P. (2019). Learning hierarchical priors in vaes. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Koller, D. and Friedman, N. (2009). In *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.
- Kondor, R. (2008). Group theoretical methods in machine learning.
- Kondor, R. (2018). N-body networks: a covariant hierarchical neural network architecture for learning atomic potentials. *CoRR*, abs/1803.01588.
- Kondor, R. and Borgwardt, K. M. (2008). The skew spectrum of graphs. 25:496–503.
- Kondor, R., Hy, T. S., Pan, H., Trivedi, S., and Anderson, B. M. (2018a). Covariant compositional networks for learning graphs. *Proc. ICLR Workshop*.
- Kondor, R. and Jebara, T. (2003). A kernel between sets of vectors. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning, ICML’03*, page 361–368. AAAI Press.
- Kondor, R., Lin, Z., and Trivedi, S. (2018b). Clebsch–gordan nets: a fully fourier space spherical convolutional neural network. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 10117–10126. Curran Associates, Inc.
- Kondor, R. and Pan, H. (2016). The multiscale Laplacian graph kernel. 29:2982–2990.
- Kondor, R., Teneva, N., and Garg, V. (2014). Multiresolution matrix factorization. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1620–1628, Bejing, China. PMLR.
- Kondor, R. and Trivedi, S. (2018). On the generalization of equivariance and convolution in neural networks to the action of compact groups. *International Conference on Machine Learning (ICML)*.
- Kondor, R. I. and Lafferty, J. D. (2002). Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the Nineteenth International Conference on Machine Learning, ICML ’02*, page 315–322, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Kostelec, P. and Rockmore, D. (2007). Soft: So(3) fourier transforms.
- Kriege, N. M., Giscard, P.-L., and Wilson, R. C. (2016). On valid optimal assignment kernels and applications to graph classification. In *Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS’16*, page 1623–1631, Red Hook, NY, USA. Curran Associates Inc.

- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. 25:1097–1105.
- Kumar, S., Mohri, M., and Talwalkar, A. (2012). Sampling methods for the nyström method. *Journal of Machine Learning Research*, 13(34):981–1006.
- Kyunghyun, C., Bart, V. M., Caglar, G., Dzmitry, B., Fethi, B., Holger, S., and Yoshua, B. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Lafferty, J. and Lebanon, G. (2005). Diffusion kernels on statistical manifolds. *Journal of Machine Learning Research*, 6(5):129–163.
- LeCun, Y., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, pages 2278–2324.
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521:436–444.
- LeCun, Y., Cortes, C., and Burges, C. J. (1999). The mnist database of handwritten digits.
- Lee, A. B., Nadler, B., and Wasserman, L. (2008). Treelets—An adaptive multi-scale basis for sparse unordered data. *The Annals of Applied Statistics*, 2(2):435 – 471.
- Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z. (2010). Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(33):985–1042.
- Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2016a). Gated graph sequence neural networks. 4.
- Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. W. (2018). Learning deep generative models of graphs. *ICML’18*, abs/1803.03324.
- Li, Y., Zemel, R., Brockschmidt, M., and Tarlow, D. (2016b). Gated graph sequence neural networks. In *Proceedings of ICLR’16*.
- Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D. K., Urtasun, R., and Zemel, R. (2019). Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Lin, Z., Khetan, A., Fanti, G., and Oh, S. (2018). Pacgan: The power of two samples in generative adversarial networks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.
- Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. (2019). Graph normalizing flows. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

- Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. (2018). Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.
- Looks, M., Herreshoff, M., Hutchins, D., and Norvig, P. (2017). Deep learning with dynamic computation graphs. *International Conference of Learning Representations (ICLR)*.
- Lovász, L. M. (2012). Large networks and graph limits. In *Colloquium Publications*.
- Ma, J., Cui, P., Kuang, K., Wang, X., and Zhu, W. (2019). Disentangled graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4212–4221. PMLR.
- Maaten, L. V. D. and Hinton, G. (2008). Visualizing data using t-sne. *J. Mach Learn. Res.*, 9:2579–2605.
- Mackay, D. J. C. (1997). Gaussian processes: A replacement for supervised neural network? *Tutorial lecture notes for NIPS 1997*.
- Maddison, C. J., Tarlow, D., and Minka, T. (2014). A* sampling. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.
- Maggioni, M. and Mahadevan, S. (2006). Fast direct policy evaluation using multiscale analysis of markov diffusion processes. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 601–608, New York, NY, USA. Association for Computing Machinery.
- Mahadevan, S. and Maggioni, M. (2005). Value function approximation with diffusion wavelets and laplacian eigenfunctions. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press.
- Mahoney, M. W. (2011). Randomized algorithms for matrices and data. *Found. Trends Mach. Learn.*, 3(2):123–224.
- Mallat, S. (1989a). Multiresolution approximations and wavelet orthonormal bases of $l^2(\mathbb{R})$. *Transactions of the American Mathematical Society*, 315:69–87.
- Mallat, S. (1989b). A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693.
- Mallat, S. (2008). *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, Inc., USA, 3rd edition.
- Manduchi, R., Perona, P., and Shy, D. (1998). Efficient deformable filter banks. *IEEE Trans. on Signal Process.*, 46:1168–1173.
- Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. (2019a). Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

- Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. (2019b). Invariant and equivariant graph networks. In *International Conference on Learning Representations*.
- Maron, H., Fetaya, E., Segol, N., and Lipman, Y. (2019c). On the universality of invariant networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4363–4371. PMLR.
- Maron, H., Litany, O., Chechik, G., and Fetaya, E. (2020). On learning sets of symmetric elements. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6734–6744. PMLR.
- Masci, J., Boscaini, D., Bronstein, M. M., and Vandergheynst, P. (2015). Geodesic convolutional neural networks on riemannian manifolds.
- Maslen, D. K. and Rockmore, D. N. (1996). Generalized ffts - a survey of some recent results. Technical report, USA.
- Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers and Operations Research*, 134:105400.
- Mercer, J. (1909). Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society A*, 209:415–446.
- Mika, S., Schölkopf, B., Smola, A., Müller, K. B., Scholz, M., and Rätsch, G. (1998). Kernel pca and de-noising in feature spaces. *Advances in Neural Information Processing Systems 11*.
- Montavon, G., Rupp, M., Gobre, V., Vazquez-Mayagoitia, A., Hansen, K., Tkatchenko, A., Müller, K.-R., and von Lilienfeld, O. A. (2013). Machine learning of molecular electronic properties in chemical compound space. *New J. Phys.*, 15.
- Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, Los Alamitos, CA, USA. IEEE Computer Society.
- Munkres, J. (1957). Trie memory. *Journal of the Society for Industrial and Applied Mathematics*, 5:32–38.
- Murphy, K. P. (2012a). Chapter 19: Undirected graphical models (markov random fields). In *Machine Learning: A Probabilistic Perspective*, pages 663–707. MIT Press.
- Murphy, K. P. (2012b). *Machine Learning: A Probabilistic Perspective*. The MIT Press.
- Neumann, M., Garnett, R., Baukhage, C., and Kersting, K. (2016). Propagation kernels: Efficient graph kernels from propagated information. *Machine Learning*, 102:209–245.

- Nielsen, M. A. and Chuang, I. L. (2010). *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.
- Niepert, M., Ahmed, M., and Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2014–2023, New York, New York, USA. PMLR.
- Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, New York, NY, USA, second edition.
- O’Connell, J., Li, Z., Hanson, J., Heffernan, R., Lyons, J., Paliwal, K., Dehzangi, I. A., Yang, Y., and Zhou, Y. (2018). Spin2: Predicting sequence profiles from protein structures using deep neural networks. *Proteins: Structure, Function, and Bioinformatics*, 86.
- Ohta, Y., Kanade, T., and Sakai, T. (1978). An analysis system for scenes containing objects with substructures. 4:752–754.
- Osborn, H. (2021). Group theory lecture notes.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. *Neural Information Processing Systems (NIPS)*.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Perona, P. (1995). Deformable kernels for early vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17:488–499.
- Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’14, page 701–710, New York, NY, USA. Association for Computing Machinery.
- Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft.
- Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.
- Ramakrishnan, R., Dral, P. O., Rupp, M., and von Lilienfeld, O. A. (2014). Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1.

- Ranganath, R., Tran, D., and Blei, D. (2016). Hierarchical variational models. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 324–333, New York, New York, USA. PMLR.
- Ravi, S. and Diao, Q. (2016). Large scale distributed semi-supervised learning using streaming approximation. *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*.
- Read, R. C. and Corneil, D. G. (1977). The graph isomorphism disease. *J. Graph Theory*, 1:339–363.
- Reddi, S. J., Kale, S., and Kumar, S. (2018). On the convergence of adam and beyond. In *International Conference on Learning Representations*.
- Riesselman, A., Shin, J.-E., Kollasch, A., McMahon, C., Simon, E., Sander, C., Manglik, A., Kruse, A., and Marks, D. (2019). Accelerating protein design using autoregressive generative models. *bioRxiv*.
- Ruddigkeit, L., van Deursen, R., Blum, L. C., and Reymond, J. (2012). Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of Chemical Information and Modeling*.
- Rue, H. and Held, L. (2005). Gaussian markov random fields: Theory and applications. In *Monographs on Statistics and Applied Probability*, volume 104, London. Chapman & Hall.
- Rupp, M., Tkatchenko, A., Müller, K. R., and von Lilienfeld, O. A. (2012). Fast and accurate modeling of molecular atomization energies with machine learning. *Phys. Rev. Lett.*, 108.
- Rustamov, R. and Guibas, L. J. (2013). Wavelets on graphs via deep learning. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.
- Sagan, B. E. (2001). *The Symmetric Group*. Grad. Texts in Math. Springer.
- Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.
- Schölkopf, B. and Smola, A. (2002). Support vector machines and kernel algorithms. *Encyclopedia of Biostatistics*, 5328-5335 (2005).
- Schölkopf, B., Smola, A. J., and Müller, K.-R. (1999). *Kernel Principal Component Analysis*, page 327–352. MIT Press, Cambridge, MA, USA.
- Schütt, K., Kindermans, P.-J., Sauceda Felix, H. E., Chmiela, S., Tkatchenko, A., and Müller, K.-R. (2017). Schnet: A continuous-filter convolutional neural network for modeling quantum interactions.
- Schütt, K. T., T., K., Arbabzadah, F., Chmiela, S., Müller, K. R., and Tkatchenko, A. (2017). Quantum-chemical insights from deep tensor neural networks. *Nat. Commun.*, 8:13890.

- Seitzer, M. (2020). pytorch-fid: FID Score for PyTorch. <https://github.com/mseitzer/pytorch-fid>. Version 0.1.1.
- Sen, P., Namata, G. M., Bilgic, M., Getoor, L., Gallagher, B., , and Eliassi-Rad, T. (2008). Collective classification in network data. *AI Magazine*, 29(3):93–106.
- Serre, J.-P. (1977). *Linear Representations of Finite Groups*, volume 42 of *Graduate Texts in Mathematics*. Springer-Verlag.
- Serviansky, H., Segol, N., Shlomi, J., Cranmer, K., Gross, E., Maron, H., and Lipman, Y. (2020). Set2graph: Learning graphs from sets. In *Advances in Neural Information Processing Systems*, volume 33, pages 22080–22091. Curran Associates, Inc.
- Shapeev, A. V. (2015). Moment Tensor Potentials: a class of systematically improvable interatomic potentials. *arXiv*.
- Shapeev, A. V. (2016). Moment tensor potentials. *Multiscale Model. Simul.*, 14(3):1153–1173.
- Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561.
- Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. (2009). Efficient graphlet kernels for large graph comparison. In van Dyk, D. and Welling, M., editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA. PMLR.
- Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. (2020). Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*.
- Shin, D., Si, S., and Dhillon, I. S. (2012). Multi-scale link prediction. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM ’12, page 215–224, New York, NY, USA. Association for Computing Machinery.
- Shor, P. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134.
- Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98.
- Si, S., Shin, D., Dhillon, I. S., and Parlett, B. N. (2014). Multi-scale spectral decomposition of massive graphs. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

- Simoncelli, E. P., Freeman, W. T., Adelson, E. H., and Heeger, D. J. (1992). Shiftable multiscale transforms. *IEEE Trans. Inf. Theory.*, 38:587–607.
- Simonovsky, M. and Komodakis, N. (2018). Graphvae: Towards generation of small graphs using variational autoencoders. *CoRR*, abs/1802.03480.
- Smith, J. S., Isayev, O., and Roitberg, A. E. (2017). Ani-1: an extensible neural network potential with dft accuracy at force field computational cost. *Chem. Sci.*, 8:3192–3203.
- Srivastava, A., Valkov, L., Russell, C., Gutmann, M. U., and Sutton, C. (2017). Veegan: Reducing mode collapse in gans using implicit variational learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Sterling, T. and Irwin, J. J. (2015). Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337. PMID: 26479676.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.
- Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press.
- Tagare, H. (2011). Notes on optimization on stiefel manifolds.
- Tang, L. and Liu, H. (2011). Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478.
- Teneva, N., Mudrakarta, P. K., and Kondor, R. (2016). Multiresolution matrix compression. In Gretton, A. and Robert, C. C., editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1441–1449, Cadiz, Spain. PMLR.
- Teo, P. C. and Hel-Or, Y. (1998). Lie generators for computing steerable functions. *Pattern Recognit. Lett.*, 16:7–17.
- Thiede, E. H., Hy, T. S., and Kondor, R. (2020). The general theory of permutation equivariant neural networks and higher order graph variational encoders. *CoRR*, abs/2004.03990.
- Thomas, N., Smidt, T., Kearnes, S. M., Yang, L., Li, L., Kohlhoff, K., and Riley, P. (2018). Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *CoRR*, abs/1802.08219.
- Toivonen, H., Srinivasan, A., King, R. D., Kramer, S., and Helma, C. (2003). Statistical evaluation of the Predictive Toxicology Challenge 2000–2001. *Bioinformatics*, 19(10):1183–1193.

- Tsaris, A., Anderson, D., Bendavid, J., Calafiura, P., Cerati, G., Esseiva, J., Farrell, S., Gray, L., Kapoor, K., Kowalkowski, J., Mudigonda, M., Prabhat, M., Spentzouris, P., Spiropoulou, M., Vlimant, J.-R., Zheng, S., and Zurawski, D. (2018). The hep.trkx project: Deep learning for particle tracking. *Journal of Physics: Conference Series*, 1085:042023.
- Tu, Z. W., Chen, X. R., Yuille, A. L., and Zhu, S. C. (2005). Image parsing: Unifying segmentation, detection, and recognition. *Int. J. Comput. Vis.*, 63:113–140.
- Vahdat, A. and Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems*, volume 33, pages 19667–19679. Curran Associates, Inc.
- van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.
- Vapnik, V., Golowich, S. E., and Smola, A. (1996). Support vector method for function approximation, regression estimation and signal processing. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS’96, page 281–287, Cambridge, MA, USA. MIT Press.
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations*.
- Vishwanathan, S. V. N. (2002). Kernel methods: Fast algorithms and real life applications. *PhD thesis, Indian Institute of Science, Bangalore, India*.
- Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. (2010). Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242.
- von Lilienfeld, O. A., Ramakrishnan, R., Rupp, M., and Knoll, A. (2015). Fourier series of atomic radial distribution functions: A molecular fingerprint for machine learning models of quantum chemical properties. *Int. J. Quantum Chem.*, 115.
- Wainwright, M. J. and Jordan, M. I. (2005). A variational principle for graphical models. *New Directions in Statistical Signal Processing*.
- Wale, N., Watson, I., and Karypis, G. (2008). Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 14:347–375.
- Weiler, M., Geiger, M., Welling, M., Boomsma, W., and Cohen, T. (2018). 3d steerable cnns: Learning rotationally equivariant features in volumetric data. *CoRR*, abs/1807.02547.
- Weisfeiler, B. and Lehman, A. A. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 9.
- Wen, Z. and Yin, W. (2010). A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142.

- Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 1168–1175, New York, NY, USA. Association for Computing Machinery.
- Williams, C. and Seeger, M. (2001). Using the nyström method to speed up kernel machines. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press.
- Williams, R. J. (1988). Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, Northeastern University, College of Computer Science.
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256.
- Worrall, D. E., Garbin, S., Turmukhambetov, D., and Brostow, G. J. (2017). Harmonic networks: Deep translation and rotation equivariance.
- Wu, H., Köhler, J., and Noe, F. (2020). Stochastic normalizing flows. In *Advances in Neural Information Processing Systems*, volume 33, pages 5933–5944. Curran Associates, Inc.
- Xu, B., Shen, H., Cao, Q., Qiu, Y., and Cheng, X. (2019). Graph wavelet neural network. In *International Conference on Learning Representations*.
- Yang, Y., Feng, Z., Song, M., and Wang, X. (2020). Factorizable graph convolutional networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 20286–20296. Curran Associates, Inc.
- Yang, Z., Cohen, W. W., and Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 40–48. JMLR.org.
- Yao, K., Herr, J. E., Toth, D., Mckintyre, R., and Parkhill, J. (2018). The tensormol-0.1 model chemistry: a neural network augmented with long-range physics. *Chem. Sci.*, 9:2261–2269.
- Yates, F. (1958). Design and analysis of factorial experiments.
- Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.
- You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. (2018a). GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5708–5717. PMLR.
- You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. (2018b). Code for graphrnn: Generating realistic graphs with deep auto-regressive model.

- Zachary, W. (1976). An information flow model for conflict and fission in small groups1. *Journal of anthropological research*, 33.
- Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Zhang, L., Han, J., Wang, H., Car, R., and E, W. (2017). Deep Potential Molecular Dynamics: a scalable model with the accuracy of quantum mechanics. *arXiv:1707.09571 [physics]*. arXiv: 1707.09571.
- Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In *AAAI*.
- Zhang, S., Ding, Z., and Cui, S. (2019). Introducing hypergraph signal processing: Theoretical foundation and practical applications. *IEEE Internet of Things Journal*, PP:1–1.
- Zhou, D., Zheng, L., Xu, J., and He, J. (2019). Misc-gan: A multi-scale generative model for graphs. *Frontiers in Big Data*, 2:3.
- Zhu, S. and Mumford, D. (2006). A stochastic grammar of images. *Found. Trends Comput. Graphics Vis.*, 2:259–362.
- Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML’03, page 912–919. AAAI Press.
- Ziletti, A., Kumar, D., Scheffler, M., and Ghiringhelli, L. (2018). Insightful classification of crystal structures using deep learning. *Nature Communications*, 9.

CHAPTER 9

APPENDIX: GROUP & REPRESENTATION THEORY

9.1 Chapter Introduction

The mathematical theory of groups and representations is useful at several points in the study of this thesis. This appendix reviews some basic material, summarizes many of the fundamental concepts and provides important definitions on group and representation theory, but do not attempt to explain very much, as it is a vast subject. In addition, this appendix offers a brief introduction to Fourier analysis and its relation to group and representation theory. Interested readers in the subjects are recommended to read (Serre, 1977), (Kondor, 2008) and (Osborn, 2021).

9.2 Groups

9.2.1 Basic definitions

Definition 9.2.1 (Group). A set G with a binary operation $\cdot : G \times G \rightarrow G$ is called a group if:

- (Closure) $\forall g, g' \in G, gg' \in G$.
- (Associativity) $(gg')g'' = g(g'g'')$ for any $g, g', g'' \in G$.
- (Identity) $\exists e \in G$ such that $eg = ge = g$ for all $g \in G$.
- (Inverses) $\forall g \in G$ there is an element $g^{-1} \in G$ such that $gg^{-1} = g^{-1}g = e$.

A group is formally defined by 4 axioms as in Def. 9.2.1. A group G is finite if the number of elements in G is finite. A group G is *Abelian* or commutative if

$$gg' = g'g, \quad \forall g, g' \in G.$$

A simple example of a finite Abelian group is the additive group \mathbb{Z}_n of integers modulo n . The order of G , denoted $|G|$ is the cardinality of G as a set. For any $g \in G$, the smallest integer m such that $g^m = e$ is the order of g . Two groups $G = \{g_i\}$ and $G' = \{g'_j\}$ are *isomorphic*, $G \simeq G'$, if there is a one to one correspondence $\theta : g_i \leftrightarrow g'_j$ between the elements consistent with the group multiplication rules. Even if $G \simeq G'$ there is not necessarily a unique choice for θ but we must have $\theta : e \leftrightarrow e'$. A crucial consequence of the basic group axioms is: $\{g_i g\} = \{g_i\}$ for any g since $g_j g = g_i g \Rightarrow g_j = g_i$. For two groups G_1 and G_2 , we define a *direct product* group $G_1 \times G_2$ formed by pairs of elements $\{(g_1, g_2)\}$, belonging to (G_1, G_2) , which is defined by the rules:

$$e = (e_1, e_2), \quad (g_1, g_2)(g'_1, g'_2) = (g_1 g'_1, g_2 g'_2), \quad (g_1, g_2)^{-1} = (g_1^{-1}, g_2^{-1}).$$

9.2.2 Subgroups & Cosets

For any group G , a *subgroup* $H \subset G$ is defined as a set of elements belonging to G which also forms a group. A *proper* subgroup H is when $H \neq G$ and is denoted as $H < G$. For any subgroup H , we define an *equivalence* relation between two elements g_i and g'_i in G :

$$g_i \sim g'_i \Leftrightarrow g_i = g'_i h \text{ for } h \in H.$$

The *left cosets* of H in G are the sets obtained by multiplying each element of H by a fixed element g of G , where g is the left factor:

$$gH \triangleq \{gh : h \in H\}.$$

The *right cosets* are defined similarly, except that the element g is now a right factor:

$$Hg \triangleq \{hg : h \in H\}.$$

In general, the left and right cosets are different. The cosets form the *coset space* G/H with dimension $\dim(G/H) = |G|/|H|$. For any subgroup $H \subset G$, Lagrange's theorem (see 9.2.1) states that $|H|$ divides $|G|$. In general, G/H is not a group since $g_i \sim g'_i$, $g_j \sim g'_j$ does not imply $g_i g_j \sim g'_i g'_j$. If g and x are elements of G , then the conjugate of x with respect to g is the element $g^{-1}xg$. The *conjugacy class* G_x of an element x in a group G is defined by:

$$G_x \triangleq \{g^{-1}xg : g \in G\}.$$

A *normal* or *invariant subgroup* is a subgroup $N \subset G$ such that:

$$gNg^{-1} = N \quad (\forall g \in G).$$

In this case, G/N becomes a group since for $g'_i = g_i h_i$ and $g'_j = g_j h_j$ with $h_i, h_j \in N$, then $g'_i g'_j = g_i g_j h$ for some $h \in N$. And G/N is called the *quotient group* or *factor group*.

Theorem 9.2.1 (Lagrange's theorem). *If H is a subgroup of a finite group G then $|H|$ divides $|G|$.*

A set of elements g_1, \dots, g_ℓ in a group G is said to *generate* the group G if every element of G can be written as a product of elements from the list g_1, \dots, g_ℓ , and we write $G = \langle g_1, \dots, g_\ell \rangle$. For example, a *cyclic group* G possesses a *generator* a such that any element $g \in G$ can be written as a^n for some integer n , and we write $G = \langle a \rangle$. The group formed by $\{e, g, g^2, \dots, g^{r-1}\}$ where r is the order of $g \in G$ is a *cyclic subgroup* $H = \langle g \rangle$ of G . In general, suppose G has size $|G|$, it is possible to show that there is a set of $\log(|G|)$ generators generating G .

9.3 Representations

9.3.1 Matrix groups & Character

Let M_d be the set of $d \times d$ complex matrices. A matrix group is a set of matrices in M_d which satisfy the properties of a group under matrix multiplication. We denote the identity element in such groups as I . A *representation* ρ of a group G is defined as a function which maps G to a matrix group, preserving group multiplication (see Def. 9.3.1). Obviously, any group G has a trivial one-dimensional representation, $\rho_{\text{trivial}}(g) = (1)$. For any representation $\rho : G \rightarrow \mathbb{C}^{d \times d}$, we have some trivial properties such as: $\rho(e) = I$ and $\rho(g^{-1}) = \rho(g)^{-1}$. Furthermore, ρ is said to be *unitary* if each $\rho(g)$ is unitary, and it is said to be *faithful* if $\rho(g_1) \neq \rho(g_2)$ unless $g_1 = g_2$. The *regular representation* is a faithful representation which exists for any group, and is constructed as in Def. 9.3.4.

A *group homomorphism* is a map between groups that preserves the group operation (see Def. 9.3.2). A *group isomorphism* from group G to group H is a bijective group homomorphism from G to H . Alternatively, we can define the representation based on homomorphism as in Def. 9.3.3.

Definition 9.3.1 (Representation). Given a group G , a d -dimensional representation of G is a matrix valued function $\rho : G \rightarrow \mathbb{C}^{d \times d}$ such that

$$\rho(x)\rho(y) = \rho(xy)$$

for all x and y in G .

Definition 9.3.2 (Homomorphism). Let G and H be two groups. A *homomorphism* $\phi : G \rightarrow H$ is a map satisfying

$$\phi(xy) = \phi(x)\phi(y)$$

for all x and y in G .

Definition 9.3.3 (Representation). Let G be a group and V be a vector space over a field \mathbb{F} . A representation of G on V is a homomorphism

$$\rho : G \rightarrow \mathrm{GL}(V),$$

where $\mathrm{GL}(V)$ is the *general linear group* on V , or the group of invertible linear maps $\phi : V \rightarrow V$.

Definition 9.3.4 (Regular representation). Let G be a finite group and let us order its elements $g_1, g_2, \dots, g_{|G|}$. Then

$$[\rho(g)]_{i,j} = \begin{cases} 1 & \text{if } g \cdot g_j = g_i \\ 0 & \text{otherwise} \end{cases}$$

is a $|G|$ -dimensional representation of G called the regular representation.

The *character* of a representation ρ of G is a function on the group defined by $\chi_\rho(g) = \mathrm{tr}(\rho(g))$, for $g \in G$, where $\mathrm{tr}(\cdot)$ is the trace function on matrices. The character has the following properties:

- $\chi_\rho(I) = n$,
- $|\chi_\rho(g)| \leq n$,
- $|\chi_\rho(g)| = n \Rightarrow \rho(g) = e^{i\theta}I$,
- $\chi_\rho(g^{-1}) = \chi_\rho^*(g)$,
- χ_ρ is constant on any given conjugacy class of G .

9.3.2 Reducible & Irreducible representations

Two representations are said to be *equivalent* if they are isomorphic, and corresponding elements under the isomorphism have the same character (see Def. 9.3.5). Reducible and irreducible representations are defined in Def. 9.3.6. Schur's lemma (see 9.3.1) is a useful property of irreducible representations. Furthermore, theorem 9.3.2 connects irreducibility with characters.

Definition 9.3.5 (Equivalent representations). Two representations ρ_1 and ρ_2 of group G are said to be *equivalent* if for some fixed invertible matrix T ,

$$\rho_1(g) = T\rho_2(g)T^{-1} \quad (\forall g \in G).$$

Definition 9.3.6 (Reducible & irreducible representations). A representation ρ of a group G is said to be *reducible* if it is equivalent to a representation ρ' of the form

$$\rho'(g) = \begin{pmatrix} \rho_{1,1}(g) & \rho_{1,2}(g) \\ \mathbf{0} & \rho_{2,2}(g) \end{pmatrix}$$

or in other words, for some fixed invertible matrix T ,

$$\rho(g) = T \begin{pmatrix} \rho_{1,1}(g) & \rho_{1,2}(g) \\ \mathbf{0} & \rho_{2,2}(g) \end{pmatrix} T^{-1}$$

for all $g \in G$. Otherwise, it is said to be *irreducible*. The representation is said to be *completely reducible* if it is equivalent to another representation which is of block diagonal form, or for some T ,

$$\rho(g) = T \begin{pmatrix} \rho_{1,1}(g) & \mathbf{0} \\ \mathbf{0} & \rho_{2,2}(g) \end{pmatrix} T^{-1}$$

for all $g \in G$.

See examples 9.3.1, 9.3.3 and 9.3.2 for irreducible representations of some useful groups.

Example 9.3.1 (Cyclic group, integers and real numbers). *The cyclic group C_n (also denoted as $\mathbb{Z}/n\mathbb{Z}$) is the group of n integers $\{0, 1, \dots, n - 1\}$ with respect to addition modulo n such as $g_a g_b = g_{(a+b) \bmod n}$. This is an Abelian group. The irreducible representations of $\mathbb{Z}/n\mathbb{Z}$ are the functions:*

$$\rho_k(g_a) = e^{2\pi i k a / n}, \quad k \in \{0, 1, \dots, n - 1\}.$$

The integers \mathbb{Z} form a group with respect to addition. The irreducible representations are:

$$\rho_k(x) = e^{-ikx}, \quad k \in [0, 2\pi).$$

The real numbers \mathbb{R} also form a group with respect to addition. The irreducible representations are:

$$\rho_k(x) = e^{2\pi i k x}, \quad k \in \mathbb{R}.$$

Example 9.3.2 (Symmetry group). \mathbb{S}_n is the group of permutations of n elements. Suppose in the case $n = 3$, we order these permutations as: 123, 231, 312, 213, 132, and 321. There are two one-dimensional irreducible representations of \mathbb{S}_3 :

- The trivial representation: $\rho_{\text{trivial}}(g) = (1)$.
- The non-trivial (i.e., based on the sign or number of inversions of each permutation): $\rho_{\text{sign}}(g) = 1, 1, 1, -1, -1, -1$ in the above order.

There exists a two-dimensional irreducible representation, with the matrices

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \frac{1}{2} \begin{pmatrix} -1 & -\sqrt{3} \\ \sqrt{3} & -1 \end{pmatrix}, \quad \frac{1}{2} \begin{pmatrix} -1 & \sqrt{3} \\ -\sqrt{3} & -1 \end{pmatrix},$$

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \frac{1}{2} \begin{pmatrix} 1 & -\sqrt{3} \\ \sqrt{3} & -1 \end{pmatrix}, \quad \frac{1}{2} \begin{pmatrix} 1 & -\sqrt{3} \\ -\sqrt{3} & -1 \end{pmatrix}.$$

Example 9.3.3 (3D rotation group). *The special orthogonal group $\mathrm{SO}(3)$ is the group of 3-dimensional orthogonal matrices with determinant +1, that is also known as the Lie group of linear transformations of \mathbb{R}^3 that preserve inner products and chirality. The irreducible representations of $\mathrm{SO}(3)$ are the Wigner D-matrices as a function of the (α, β, γ) Euler angles:*

$$D_{m,m'}^\ell(\alpha, \beta, \gamma) = \sqrt{\frac{4\pi}{2\ell+1}} (Y_m^\ell(\beta, \alpha))^* e^{-im'\gamma},$$

where $Y_m^\ell(\beta, \alpha)$ denotes the spherical harmonics.

Theorem 9.3.1 (Schur's Lemma). *Let $\rho : G \rightarrow GL(V)$ and $\rho' : G \rightarrow GL(W)$ be two irreducible representations of a group G over \mathbb{C} and ϕ be a bounded linear map $V \rightarrow W$ such that $\phi(\rho(g)v) = \rho'(g)(\phi(v))$ for all $v \in V$. Then*

- If ρ_1 and ρ_2 are not equivalent, then $\phi = 0$.
- If $\rho_1 = \rho_2$, then $\phi = \alpha I$ for some $\alpha \in \mathbb{C}$.

Theorem 9.3.2. *A representation ρ of G is irreducible if and only if*

$$\frac{1}{|G|} \sum_{g \in G} |\chi_\rho(g)|^2 = 1.$$

The key theorem of representation theory is the following **Fundamental Theorem** (see 9.3.3).

Theorem 9.3.3 (Fundamental Theorem). *Every group G has exactly r inequivalent irreducible representations, where r is the number of conjugacy classes of G . And if $\rho^p \in M_{d_\rho}$*

and ρ^q are any two of these, then the matrix elements satisfy the orthogonality relations

$$\sum_{g \in G} [\rho^p(g)]_{ij}^{-1} [\rho^q(g)]_{kl} = \frac{|G|}{d_\rho} \delta_{il} \delta_{jk} \delta_{pq},$$

where the Kronecker delta $\delta_{pq} = 1$ if $\rho^p = \rho^q$ and is zero otherwise.

From the Fundamental theorem, we can see that characters are orthogonal, that is

$$\sum_{i=1}^r r_i (\chi_i^p)^* \chi_i^q = |G| \delta_{pq}$$

and

$$\sum_{p=1}^r (\chi_i^p)^* \chi_j^p = \frac{|G|}{r_i} \delta_{ij},$$

where p , q , and δ_{pq} have the same meaning as in the theorem, and χ_i^p is the value the character of the p -th irreducible representation takes on the i -th conjugacy class of G , and r_i is the size of the i -th conjugacy class. The decomposition of arbitrary representations into tensor sums of irreducible representations obeys theorem 9.3.4.

Theorem 9.3.4. *If ρ is an arbitrary representation of G with character χ , and ρ^p are the inequivalent irreducible representations of G with character χ^p , then*

$$\rho = \bigoplus_p c_p \rho^p,$$

where \oplus denotes a direct sum, and c_p are the numbers determined by

$$c_p = \frac{1}{|G|} \sum_{i=1}^r r_i (\chi_i^p)^* \chi_i.$$

Let \mathcal{R} denote the set of all inequivalent irreducible representations. We have the following

identity:

$$\sum_{\rho \in \mathcal{R}} d_\rho^2 = |G|, \quad (9.1)$$

where ρ is a matrix group of $d_\rho \times d_\rho$ matrices.

9.4 Fourier Transforms On Groups

In chapter 6 – Learning Multiresolution Matrix Factorization and its Wavelet Networks on Graphs, we have discussed the Graph Fourier Transform (GFT) and our Graph Wavelet Transform (GWT). In this appendix, we will discuss the classical Fourier transforms and the generalized Fourier transforms on finite groups. Readers interested in efficient computation of Fourier transforms over groups are recommended to read (Diaconis and Rockmore, 1990), (Clausen and Baum, 1993), (Maslen and Rockmore, 1996), and (Kondor, 2008).

9.4.1 Classical Fourier Transform

The classical Fourier transforms, or sometimes called Fourier series, are defined on the unit circle or periodic functions on the real line with period 2π ,

$$\hat{f}(k) = \frac{1}{2\pi} \int_0^{2\pi} e^{-ikx} f(x) dx, \quad k \in \mathbb{Z};$$

and functions on the real line,

$$\hat{f}(k) = \int_{-\infty}^{\infty} e^{-2\pi ikx} f(x) dx, \quad k \in \mathbb{R}.$$

We assume that f might take complex values. The corresponding inverse transforms are

$$f(x) = \sum_{k=-\infty}^{\infty} \hat{f}(k) e^{ikx},$$

and

$$f(x) = \int_{-\infty}^{\infty} e^{2\pi i k x} \hat{f}(k) dk,$$

respectively. We use the notation $\mathcal{F} : f \mapsto \hat{f}$ for the forward transform and $\mathcal{F}^{-1} : \hat{f} \mapsto f$ for the inverse transform. The *Parseval's theorem* or *Plancherel's theorem* states that all these transforms are unitary, i.e., inner product preserving $\langle f, g \rangle = \langle \hat{f}, \hat{g} \rangle$ and consequentially $\|f\|^2 = \|\hat{f}\|^2$. With respect to the appropriate inner products, we have:

$$\langle f, g \rangle = \frac{1}{2\pi} \int_0^{2\pi} f(x) g(x)^* dx, \quad \langle \hat{f}, \hat{g} \rangle = \sum_{k=-\infty}^{\infty} \hat{f}(k) \hat{g}(k)^*,$$

and

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x) g(x)^* dx, \quad \langle \hat{f}, \hat{g} \rangle = \int_{-\infty}^{\infty} \hat{f}(k) \hat{g}(k)^* dk,$$

respectively. Furthermore, the *convolution theorem* states that the Fourier transform of a convolution of two functions is the pointwise product of their Fourier transforms, i.e.,

$$\mathcal{F}(f * g)(k) = \hat{f}(k) \cdot \hat{g}(k).$$

We will continue to discuss the computational algorithms such as Discrete Fourier Transforms and Fast Fourier Transforms as follows.

Discrete Fourier Transform 1D

Discrete Fourier Transform (DFT) of 1-dimensional real signal $\mathbf{f} = (f_0, \dots, f_{N-1}) \in \mathbb{R}^N$ is defined as

$$\hat{f}_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_n \cdot e^{-2\pi i k \frac{n}{N}}.$$

On the another hand, the inverse transform is

$$f_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \hat{f}_k \cdot e^{2\pi i k \frac{n}{N}}.$$

DFT can be expressed as a matrix multiplication $\hat{\mathbf{f}} = \mathbf{W}\mathbf{f}$ where $\mathbf{W} = \left(\frac{w^{jk}}{\sqrt{N}} \right)_{j,k=0,\dots,N-1}$ is the $N \times N$ DFT matrix, or equivalently

$$\mathbf{W} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^{N-1} \\ 1 & w^2 & w^4 & \cdots & w^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{N-1} & w^{2(N-1)} & \cdots & w^{(N-1)(N-1)} \end{pmatrix},$$

where $w = e^{-2\pi i/N}$ is a primitive N th root of unity (i.e. $z^N = 1$). In the group theoretic perspective, the N th roots of unity form an *irreducible representation* of cyclic group of order N . Indeed, DFT can be interpreted as Fourier transform on group $\mathbb{Z}/N\mathbb{Z}$ whose characters are functions $\rho_0, \dots, \rho_{N-1}$ defined by $\rho_k(j) = w^{jk}$:

$$\hat{f}_k = \hat{f}(\rho_k) = \sum_{j \in \mathbb{Z}/N\mathbb{Z}} \rho_k(j) f_j.$$

We will discuss more about Fourier transforms on finite groups later in this chapter. The time complexity of DFT is $O(N^2)$. The Fourier coefficient at the k -th frequency with the corresponding real and complex parts:

$$\text{Re}(\hat{f}_k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_n \cdot \cos\left(2\pi k \frac{n}{N}\right),$$

$$\text{Im}(\hat{f}_k) = -\frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_n \cdot \sin\left(2\pi k \frac{n}{N}\right).$$

The amplitude and the phase are, respectively:

$$|\hat{f}_k| = \sqrt{\text{Re}(\hat{f}_k)^2 + \text{Im}(\hat{f}_k)^2},$$

$$\arg(\hat{f}_k) = \text{atan2}(\text{Im}(\hat{f}_k), \text{Re}(\hat{f}_k)).$$

Discrete Fourier Transform 2D

DFT of 2-dimensional signal $f \in \mathbb{R}^{M \times N}$ is defined as

$$\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f_{u,v} \cdot e^{-2\pi i \left(\frac{xu}{M} + \frac{yv}{N} \right)},$$

or

$$\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} e^{-2\pi i \frac{xu}{M}} \cdot \left(\sum_{v=0}^{N-1} f_{u,v} \cdot e^{-2\pi i \frac{yv}{N}} \right).$$

Let $P(u, y) = \sum_{v=0}^{N-1} f_{u,v} \cdot e^{-2\pi i \frac{yv}{N}}$. Then, we get

$$\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} e^{-2\pi i \frac{xu}{M}} \cdot P(u, y).$$

The inverse Fourier transform is

$$f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}_{x,y} \cdot e^{2\pi i \left(\frac{xu}{M} + \frac{yv}{N} \right)},$$

or

$$f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} e^{2\pi i \frac{xu}{M}} \cdot \left(\sum_{y=0}^{N-1} \hat{f}_{x,y} \cdot e^{2\pi i \frac{yv}{N}} \right).$$

Let $H(x, v) = \sum_{y=0}^{N-1} \hat{f}_{x,y} \cdot e^{2\pi i \frac{yv}{N}}$. Then, we get

$$f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} e^{2\pi i \frac{xu}{M}} \cdot H(x, v).$$

The Fourier image (to visualize amplitudes or phases) is shifted in such a way that the value (image mean) $\hat{f}(0, 0)$ is displayed in the center of the image. The further away from the center an image point is, the higher is its corresponding frequency. The dynamic range of

the Fourier coefficients (for example, the intensity values in the Fourier image) is too large to be displayed on the screen, therefore all other values appear as black. We need to apply a logarithmic transformation to the image as follows:

$$Q(i, j) = c \log(1 + |P(i, j)|)$$

where $P(i, j)$ is the original image, and $Q(i, j)$ is the transformed image. The scaling constant c is chosen so that the maximum output value is 255 (providing an 8-bit format). That means if R is the value with the maximum magnitude in the input image, c is given by:

$$c = \frac{255}{\log(1 + |R|)}.$$

Discrete Cosine Transform 2D

Indeed, Discrete Cosine Transform (DCT) is a special case of DFT:

$$\hat{f}_{u,v} = \frac{1}{4} \alpha(u) \alpha(v) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f_{x,y} \cos \left[\frac{(2x+1)u\pi}{2M} \right] \cos \left[\frac{(2y+1)v\pi}{2N} \right]$$

where $\alpha(u) = \frac{1}{\sqrt{2}}$ if $u = 0$, and $\alpha(u) = 1$ otherwise. The inverse DCT is:

$$f_{x,y} = \frac{1}{4} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \hat{f}_{u,v} \cdot \alpha(u) \alpha(v) \cos \left[\frac{(2x+1)u\pi}{2M} \right] \cos \left[\frac{(2y+1)v\pi}{2N} \right].$$

For image compression such as JPEG, DCT is applied for small batches of 8×8 (see Fig. 9.1).

Fast Fourier Transform 1D

Historically, Cooley and Tukey were interested in the analysis of time series. The particular application that Cooley and Tukey had in mind early in the 1960's was the analysis of seismic data. At this time, a nuclear test with the (then) USSR was under negotiation. The USSR was balking at the notion of site visits, so it was necessary that there be some way

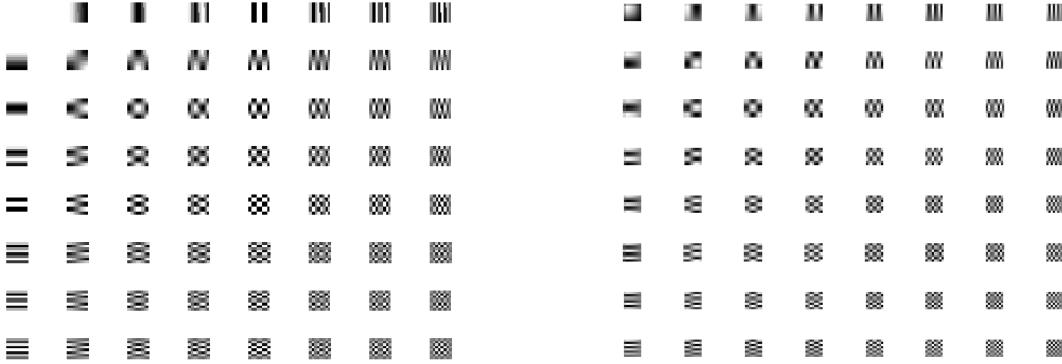


Figure 9.1: The left is the DCT basis functions used for JPEG image compression, and the right is the basis functions of the inverse DCT. My source code is available at <https://github.com/HyTruongSon/Fourier-Transform-Library>.

of remotely confirming compliance. The prevailing idea was to surround the Soviet Union with many sensors in order to monitor seismic activity. Nuclear detonations could then be detected by particular structure in the Fourier transforms of the collection of time series.

Cooley-Tukey FFT algorithm (Cooley and Tukey, 1965) was first invented by Carl Friedrich Gauss and then rediscovered by Cooley and Tukey (the radix-2 Decimation In Time or DIT case). DFT is defined by the formula:

$$\hat{f}_k = \sum_{n=0}^{N-1} f_n \cdot e^{-\frac{2\pi i}{N} nk}, \quad k = 0, 1, \dots, N-1$$

Radix-2 DIT first computes the DFTs of the even-indexed inputs ($f_{2m} = f_0, f_2, \dots, f_{N-2}$) and of the odd-indexed inputs ($f_{2m+1} = f_1, f_3, \dots, f_{N-1}$) and then combines those two results to produce the DFT of the whole sequence:

$$\hat{f}_k = \sum_{m=0}^{N/2-1} f_{2m} \cdot e^{-\frac{2\pi i}{N} (2m)k} + \sum_{m=0}^{N/2-1} f_{2m+1} \cdot e^{-\frac{2\pi i}{N} (2m+1)k}$$

$$\Leftrightarrow \hat{f}_k = \sum_{m=0}^{N/2-1} f_{2m} \cdot e^{-\frac{2\pi i}{N/2} mk} + e^{-\frac{2\pi i}{N} k} \sum_{m=0}^{N/2-1} f_{2m+1} \cdot e^{-\frac{2\pi i}{N/2} mk},$$

that can be written as

$$\hat{f}_k = E_k + e^{-\frac{2\pi i}{N} k} \cdot O_k,$$

where E_k is the DFT of the even-indexed part of f_m , and O_k is the DFT of the odd-indexed part of f_m . Based on the periodicity of the DFT: $E_{k+\frac{N}{2}} = E_k$, $O_{k+\frac{N}{2}} = O_k$. We have the following:

$$\hat{f}_k = \begin{cases} E_k + e^{-\frac{2\pi i}{N} k} \cdot O_k & \text{if } 0 \leq k < \frac{N}{2}, \\ E_{k-\frac{N}{2}} + e^{-\frac{2\pi i}{N} k} \cdot O_{k-\frac{N}{2}} & \text{if } \frac{N}{2} \leq k < N. \end{cases}$$

The harmonic factor $e^{-2\pi i k/N}$ have the property that:

$$e^{-\frac{2\pi i}{N}(k+\frac{N}{2})} = e^{\frac{-2\pi i k}{N} - \pi i} = -e^{-\frac{2\pi i k}{N}}.$$

Thus, we can cut the number of harmonic factor calculations in half. For $0 \leq k < \frac{N}{2}$:

$$\hat{f}_k = E_k + e^{-\frac{2\pi i}{N} k} \cdot O_k, \quad \hat{f}_{k+\frac{N}{2}} = E_k - e^{-\frac{2\pi i}{N} k} \cdot O_k.$$

The pseudocode for Cooley-Tukey FFT algorithm is detailed in Algorithm 18. The time complexity of FFT is $O(N \log_2 N)$.

Fast Fourier Transform 2D

We have the DFT 2D as

$$\begin{aligned} \hat{f}_{x,y} &= \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f_{u,v} \cdot e^{-2\pi i \left(\frac{xu}{M} + \frac{yv}{N} \right)} \\ \Leftrightarrow \hat{f}_{x,y} &= \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} e^{-2\pi i \frac{xu}{M}} \cdot \left(\sum_{v=0}^{N-1} f_{u,v} \cdot e^{-2\pi i \frac{yv}{N}} \right). \end{aligned}$$

Algorithm 18 Cooley–Tukey FFT algorithm.

This is the recursive call of $\text{FFT}(f, N, s)$ to return the FFT of $(f_0, f_s, f_{2s}, \dots)$. Here, f is an 1-dimensional array of the signals, N is the number of signals, and s is an integer. **Note:** $f + s$ (in the context of C++ programming language) means moving the array pointer of f to the s -th element.

```

if  $N = 1$  then
     $\hat{f}_0 \leftarrow f_0$ 
else
     $\hat{f}_{0,\dots,N/2-1} \leftarrow \text{FFT}(f, N/2, 2s)$ 
     $\hat{f}_{N/2,\dots,N-1} \leftarrow \text{FFT}(f + s, N/2, 2s)$ 
    for  $k = 0 \rightarrow N/2 - 1$  do
         $t \leftarrow \hat{f}_k$ 
         $\hat{f}_k \leftarrow t + \exp(-2\pi ik/N) \cdot \hat{f}_{k+N/2}$ 
         $\hat{f}_{k+N/2} \leftarrow t - \exp(-2\pi ik/N) \cdot \hat{f}_{k+N/2}$ 
    end for
end if
```

We can compute $P(u, y) = \sum_{v=0}^{N-1} f_{u,v} \cdot e^{-2\pi i \frac{yv}{N}}$ by FFT-1D. Again, we use FFT-1D to compute

$\hat{f}_{x,y} = \frac{1}{\sqrt{MN}} \sum_{u=0}^{M-1} e^{-2\pi i \frac{xu}{M}} \cdot P(u, y)$. On the another hand, the inverse pass:

$$f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}_{x,y} \cdot e^{2\pi i \left(\frac{xu}{M} + \frac{yv}{N} \right)}$$

$$\Leftrightarrow f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} e^{2\pi i \frac{xu}{M}} \cdot \left(\sum_{y=0}^{N-1} \hat{f}_{x,y} \cdot e^{2\pi i \frac{yv}{N}} \right).$$

We compute $H(x, v) = \sum_{y=0}^{N-1} \hat{f}_{x,y} \cdot e^{2\pi i \frac{yv}{N}}$ and then $f_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} e^{i2\pi \frac{xu}{M}} \cdot H(x, v)$ by FFT-1D algorithm.

My library for FFTs in MATLAB/C++ is available at

<https://github.com/HyTruongSon/Fourier-Transform-Library>

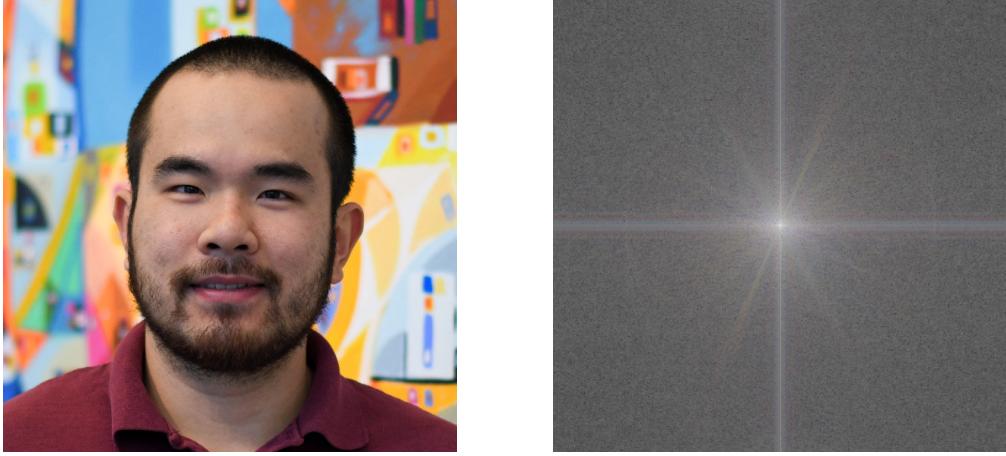


Figure 9.2: My picture and its 2D Fast Fourier Transform.

9.4.2 Generalized Fourier transform on finite groups

Let G be a finite group of order $N = |G|$, and $f : G \rightarrow \mathbb{C}$ be a function which maps group elements to complex numbers. For an irreducible representation ρ of G , of dimension d_ρ , we define the Fourier transform of f to \hat{f} as

$$\hat{f}(\rho) \triangleq \sqrt{\frac{d_\rho}{N}} \sum_{g \in G} f(g)\rho(g), \quad (9.2)$$

where $\hat{f}(\rho)$ maps matrices to matrices while ρ is a matrix representation. Let $\mathcal{R} = \{\rho^{(1)}, \dots, \rho^{(r)}\}$ be a complete set of inequivalent irreducible representations of G . Then the corresponding collection of matrix coefficients $\{\rho_{jk}^{(i)} | 1 \leq i \leq r; 1 \leq j, k \leq d_{\rho^{(i)}}\}$ form an orthogonal basis for the $|G|$ dimensional vector space of complex valued functions on G , denoted as $\mathbb{C}G$. A Fourier transform for a finite group G is a *change of basis* from the basis of point mass or delta functions for $\mathbb{C}G$ to a *basis of irreducible matrix coefficients*. We define the inverse Fourier transform of \hat{f} to be

$$f(g) \triangleq \frac{1}{\sqrt{N}} \sum_{\rho \in \mathcal{R}} \sqrt{d_\rho} \text{tr}(\hat{f}(\rho)\rho(g^{-1})). \quad (9.3)$$

Because $\sum_\rho d_\rho^2 = |G|$ (see Eq. 9.1), f and \hat{f} can be expressed as vectors of complex numbers of length N . If \mathcal{R} consists of unitary representations, then the Fourier transformation $\mathcal{F} : f \mapsto \hat{f}$ is unitary with respect to the norms

$$\|f\|^2 = \frac{1}{N} \sum_{g \in G} |f(g)|^2,$$

and

$$\|\hat{f}\|^2 = \frac{1}{N^2} \sum_{\rho \in \mathcal{R}} d_\rho \|\hat{f}(\rho)\|_F^2,$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. Substituting Eq. 9.2 into Eq. 9.3, we get

$$f(g) = \frac{1}{N} \sum_{\rho \in \mathcal{R}} \sum_{g' \in G} d_\rho f(g') \text{tr}(\rho(g')\rho(g^{-1}))$$

that is equivalent to

$$f(g) = \frac{1}{N} \sum_{\rho \in \mathcal{R}} \sum_{g' \in G} d_\rho f(g') \text{tr}(\rho(g'g^{-1})) = \frac{1}{N} \sum_{g' \in G} f(g') \sum_{\rho \in \mathcal{R}} d_\rho \chi_\rho(g'g^{-1}). \quad (9.4)$$

Given that $\sum_{\rho \in \mathcal{R}} d_\rho \chi_\rho(g) = N\delta_{ge}$ (because the character of the regular representation is zero except for the conjugacy class i containing e , the identity element in G), from Eq. 9.4 we get

$$f(g) = \sum_{g' \in G} f(g') \delta_{g'g},$$

as expected. The *convolution* of two functions $f, f' : G \rightarrow \mathbb{C}$ is defined as

$$(f * f')(a) = \sum_{b \in G} f(ab^{-1})f'(b).$$

The Fourier transform of a convolution at any representation ρ of G is given by

$$\widehat{f * f'}(\rho) = \hat{f}(\rho) \cdot \hat{f}'(\rho),$$

where \cdot denotes the matrix dot product. The Plancherel formula states that

$$\sum_{a \in G} f(a^{-1})f'(a) = \frac{1}{N} \sum_{\rho \in \mathcal{R}} d_\rho \text{tr}(\hat{f}(\rho)\hat{f}'(\rho)).$$

Example shows the Fourier transform and its inverse on the symmetry group, group of permutations of n elements, \mathbb{S}_n (Clausen and Baum, 1993).

Example 9.4.1 (Fourier transform on symmetry group). *The Fourier transform of a general function $f : \mathbb{S}_n \rightarrow \mathbb{C}$ is the collection of matrices*

$$\hat{f}(\lambda) = \sum_{\sigma \in \mathbb{S}_n} f(\sigma)\rho_\lambda(\sigma)$$

where λ extends over the integer partitions of n , and $\rho_\lambda : \mathbb{S}_n \rightarrow \mathbb{C}^{d_\lambda \times d_\lambda}$ is the corresponding irreducible representation (irrep) of \mathbb{S}_n , given in Young's Orthogonal Representation. The inverse transform is given by

$$f(\sigma) = \frac{1}{n!} \sum_{\lambda \vdash n} d_\lambda \text{tr}[\rho_\lambda^{-1}(\sigma)\hat{f}(\lambda)].$$

9.4.3 Fast Fourier Transform for group $(\mathbb{Z}/2\mathbb{Z})^k$

One version of the abelian FFT is due to the statistician and design theorist Yates (Yates, 1958). To efficiently compute the interaction analysis for data from a 2^k -factorial design, Yates described an algorithm which is an FFT for the group $(\mathbb{Z}/2\mathbb{Z})^k$ (Maslen and Rockmore, 1996). A 2^k -factorial design is the set of all k -tuples of signs $\{+1, -1\}^k$, which can be thought of as the vertices of the k -dimensional hypercube or the space of binary k -tuples. It is a natural way to index the trials of an experiment which depends on k factors, each of which maybe a set at a *high* or *low* level.

For example, we are given a dataset of the average heights of plants, denoted by α_{swf}

for a given choice of sunlight (s), weed killer (w) and fertilizer (f). This is a 2^3 -factorial design in which the data vector $\boldsymbol{\alpha}$ is considered as an element in the vector space of complex-valued functions on the group $(\mathbb{Z}/2\mathbb{Z})^3$. The zeroth order effect is the grand mean or total average height:

$$\mu_{gr} = \frac{1}{8} \sum_{(s,w,f) \in \{+, -\}^3} \alpha_{swf}.$$

The first order effects are the effect of one particular factor, while all other factors being held equal. Here is the difference (i.e., first order effect) of the average yields at a high level of sunlight versus the average at a low level of sun light:

$$\mu_S = \frac{1}{4}(\alpha_{+--} + \alpha_{+-+} + \alpha_{++-} + \alpha_{+++}) - \frac{1}{4}(\alpha_{---} + \alpha_{--+} + \alpha_{-+-} + \alpha_{-++}).$$

The collection of first, second and third order effects can be coded up as the computation of a matrix-vector multiplication (the matrix of size $2^k \times 2^k$):

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} \alpha_{+++} \\ \alpha_{-++} \\ \alpha_{+-+} \\ \alpha_{--+} \\ \alpha_{++-} \\ \alpha_{-+-} \\ \alpha_{+--} \\ \alpha_{---} \end{pmatrix} = \begin{pmatrix} 8\mu_{gr} \\ 4\mu_S \\ 4\mu_W \\ 4\mu_{SW} \\ 4\mu_F \\ 4\mu_{SF} \\ 4\mu_{WF} \\ 4\mu_{SWF} \end{pmatrix}. \quad (9.5)$$

Initially, the data is expressed in terms of the basis of delta functions on the group,

$$\boldsymbol{\alpha} = \sum_{g \in (\mathbb{Z}/2\mathbb{Z})^3} \alpha(g) \delta_g.$$

The analysis of the data is the same as computing the *projection* of the data vector α onto an *orthogonal basis* for which the coordinates seem to carry more information(see Eq. 9.5). In this case, the new basis if the basis of characters (one-dimensional representations) of group $(\mathbb{Z}/2\mathbb{Z})^3$, and the computation of the matrix vector product in Eq. 9.5 is indeed a Fourier transform. Naively, 8^2 , or $O(2^{2k})$ in general, operations are required to compute the full analysis. However, from the group theoretic perspective, we can obtain a much faster algorithm. Let H_k denote the matrix of the Fourier transform on $(\mathbb{Z}/2\mathbb{Z})^k$. We have

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

and H_3 is the matrix in Eq. 9.5. Any character of $(\mathbb{Z}/2\mathbb{Z})^3$ can be written as a tensor product of characters of the group $\mathbb{Z}/2\mathbb{Z}$. Therefore, H_3 has the factorization

$$H_3 = H_1 \otimes H_1 \otimes H_1 = [I_4 \otimes H_1] \cdot [I_2 \otimes H_1 \otimes I_2] \cdot [H_1 \otimes I_4],$$

where I_j denotes the $j \times j$ identify matrix, \otimes denotes the tensor product and \cdot denotes the matrix multiplication. This is a *sparse decomposition* of the matrix H_3 , and the Fourier transform of α is computed by multiplying by each of these sparse matrices in turn. In this case, the number of operations is reduced to $O(2^k \log(2^k))$. The transform is also known as the *Walsh-Hadamard* transform. In the group theoretic perspective, this transform is based on *factoring representations as tensor products*.

9.4.4 Group theoretic interpretation of the FFT

In section 9.4.1, we have discussed the Cooley-Tukey FFT algorithm for the DFT,

$$\hat{f}_k = \sum_{j=0}^{N-1} f_j w^{jk}, \quad k = 0, 1, \dots, N-1; \quad \text{for } w = e^{-2\pi i/N} \quad (9.6)$$

for $N = 2^n$ complex numbers $\mathbf{f} = (f_0, f_1, \dots, f_{N-1})$. In general, given a prime factorization $N = p_1 \cdots p_r$, FFT only needs $N \sum_i p_i$ operations. If each $p_i = 2$ (i.e. N is a power of 2), this is an $O(N \log_2 N)$ algorithm as detailed in section 9.4.1.

Let consider the factorization $N = pq$. We change the indexing:

$$j = i_2 + i_1 q, \quad k = m_1 + m_2 p.$$

We fine the two-dimensional arrays:

$$f_{i_1, i_2} = f_j \quad i_1 = 0, \dots, p-1 \quad i_2 = 0, \dots, q-1$$

and

$$\hat{f}_{m_1, m_2} = \hat{f}_k \quad m_1 = 0, \dots, p-1 \quad m_2 = 0, \dots, q-1.$$

Substituting into Eq. 9.6, we obtain

$$\hat{f}_{m_1, m_2} = \sum_{i_2=0}^{q-1} w^{i_2(m_1+m_2p)} \sum_{i_1=0}^{p-1} (w^q)^{i_1 m_1} f_{i_1, i_2}. \quad (9.7)$$

The computation is then performed into two steps:

- First, q transforms of length p are computed according to

$$\bar{f}_{i_2, m_1} = \sum_{i_1=0}^{p-1} (w^q)^{i_1 m_1} f_{i_1, i_2}$$

- Next, p transforms of length q are computed according to

$$\hat{f}_{m_1, m_2} = \sum_{i_2=0}^{q-1} w^{i_2(m_1+m_2p)} \bar{f}_{i_2, m_1}$$

Instead of $(pq)^2$ operations, the above uses $(pq)(p+q)$ operations. The main idea is that we have converted/factorized a one-dimensional algorithm, in terms of indexing, into a two-dimensional algorithm.

Now, we will examine this factorization in the light of group and representation theory. Let $G = \mathbb{Z}/N\mathbb{Z}$ be the group of integers modulo N with the characters $\rho_0, \dots, \rho_{N-1}$ defined as $\rho_k(j) = w^{jk}$. As mentioned in section 9.4.1, we view the sequence \mathbf{f} as a function on $\mathbb{Z}/N\mathbb{Z}$, and the sequence $\hat{\mathbf{f}}$ as a function on the group of characters, then DFT is the Fourier transform on $\mathbb{Z}/N\mathbb{Z}$:

$$\hat{f}_k = f(\rho_k) = \sum_{j \in \mathbb{Z}/N\mathbb{Z}} \rho_k(j) f_j.$$

If $N = pq$, we consider the subgroup $q\mathbb{Z}/N\mathbb{Z}$, generated by the group element q . The reindexing of group element $j = i_2 + (i_1 q)$ can be viewed as a factorization of j into the sum of group elements i_2 and $i_1 q$, in which the elements $i_1 q$ are in the subgroup $q\mathbb{Z}/N\mathbb{Z}$, and the elements i_1 form a complete set of coset representatives for $\mathbb{Z}/N\mathbb{Z}$ relative to this subgroup. The restriction of ρ_k to $q\mathbb{Z}/N\mathbb{Z}$ is the character $\rho_{k \downarrow q\mathbb{Z}/N\mathbb{Z}} = \chi_{m_1}$ where $\chi_{m_1}(i_1 q) = (w^q)^{i_1 m_1}$. The quantity \bar{f}_{i_2, m_1} is therefore indexed by pairs consisting of a coset representative for $(\mathbb{Z}/N\mathbb{Z})/(q\mathbb{Z}/N\mathbb{Z})$ and a character of $q\mathbb{Z}/N\mathbb{Z}$. The expression 9.7 can be rewritten as follows

$$\hat{f}(\rho_k) = \sum_{a \in A} \rho_k(a) \sum_{b \in q\mathbb{Z}/N\mathbb{Z}} \rho_{k \downarrow q\mathbb{Z}/N\mathbb{Z}}(b) f_{a+b},$$

where A denotes the set of coset representatives.

In conclusion, the FFT algorithm generalizes to any finite group with a proper nontrivial subgroup. In this case of $\mathbb{Z}/N\mathbb{Z}$, the reindexing scheme for characters is described as the restrictions of the irreducible representations from group to subgroup within the chain of subgroups, $\mathbb{Z}/N\mathbb{Z} > q\mathbb{Z}/N\mathbb{Z} \simeq \mathbb{Z}/p\mathbb{Z} > 1$.

9.4.5 Generalized FFTs

In general, the method of constructing fast Fourier transform algorithms on arbitrary finite groups are based on the *matrix separation of variables* approach, which generalizes the Cooley–Tukey method for cyclic groups. The basic idea is to re-index the calculation so as to replace the single sum $\hat{f}(\rho) = \sum_{g \in G} f(g)\rho(g)$ defining a Fourier transform by a multiple sum over many different coordinates (Maslen and Rockmore, 1996). Assume that each group element g has a factorization of the form $g = a_n \cdots a_1$. We substitute this into the definition and use the homomorphism property of group representations to obtain

$$\hat{f}(\rho) = \sum_{g=a_n \cdots a_1} \rho(a_n \cdots a_1) f(a_n \cdots a_1) = \sum_{g=a_n \cdots a_1} \rho(a_n) \cdots \rho(a_1) f(a_n \cdots a_1)$$

or equivalently,

$$\hat{f}(\rho) = \sum_{a_n} \rho(a_n) \cdots \sum_{a_2} \rho(a_2) \sum_{a_1} \rho(a_1) f(a_n \cdots a_1).$$

The transform is now in a multi-dimensional form, and can be computed by summing on a_1 first, and then summing on a_2 , and so on. A famous example is the Clausen’s algorithm (Clausen and Baum, 1993) constructing the fast Fourier transform on the symmetric group that follows the separation of variables idea tailored to the chain of subgroups

$$\mathbb{S}_1 < \mathbb{S}_2 < \cdots < \mathbb{S}_{n-1} < \mathbb{S}_n.$$

The essential ingredients of the matrix separation of variables approach are the factorization of group elements and the use of adapted sets of representations, or Gel’fand-Tsetlin bases. Readers interested in this topic are recommended to read (Diaconis and Rockmore, 1990), (Clausen and Baum, 1993), (Maslen and Rockmore, 1996), and (Kondor, 2008).

9.4.6 Quantum Fourier Transform

Quantum computing and *quantum information* is the study of the information processing tasks that can be accomplished using quantum mechanical systems. The most spectacular discovery in quantum computing to date is that quantum computers can efficiently perform some tasks which are not feasible on a classical computers. For instance, a quantum computer can factor an integer exponentially faster than the best known classical algorithms. In particular, (Shor, 1994) proposed *Shor's algorithm*, a quantum computer algorithm for finding the prime factors of an n -bit integer in polynomial time $O(n^2 \log n \log \log n)$. The mathematical theory of groups and representations is useful in the study of quantum computation and quantum information. For example, the generalization of the order-finding, factoring, and period finding algorithms is based on the *hidden subgroup problem*¹; and the quantum circuits are implicitly an example of using Lie groups. The key ingredient for many important quantum algorithms such as phase estimation and order-finding is the *Quantum Fourier Transform* (QFT). With such a motivation, this introduction about QFT is placed in the appendix chapter of group & representation theory and in the section of generalized Fourier transform on groups.

In classical computing and classical information, the *bit* or *classical bit* is a fundamental concept. A classical bit has a *state* that can be either 0 or 1. In quantum computing and quantum information, we have an analogous concept that is *quantum bit* or *qubit* for short. Two possible states for a qubit are the states $|0\rangle$ and $|1\rangle$, where $|\cdot\rangle$ is the standard *Dirac notation* for states in quantum mechanics. The fundamental difference between bits and

1. Given a group G , a subgroup $H \leq G$, and a finite set X . We say a function $f : G \rightarrow X$ hides the subgroup H if for all $g_1, g_2 \in G$, $f(g_1) = f(g_2)$ if and only if $g_1H = g_2H$. Equivalently, the function f is constant on the cosets of H , while it is different between the different cosets of H . The hidden subgroup problem (HSP) assumes that the function f is given via an oracle, which uses $O(\log |G| + \log |X|)$ bits. Using information gained from evaluations of f via its oracle, we need to determine a generating set for H . HSP is a generalization of the phase estimation and order-finding problems, and can be solved by using Quantum Fourier Transform.

qubits is that a qubit can be in a state *other* than two special $|0\rangle$ or $|1\rangle$ states, that we call *superpositions* (i.e. similar to quantum mechanics) defined as a linear combination of states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where $\alpha, \beta \in \mathbb{C}$ satisfying the normalization constraint $|\alpha|^2 + |\beta|^2 = 1$. In other words, the state of a qubit is a vector in \mathbb{C}^2 , while $|0\rangle$ and $|1\rangle$ are known as *computational basis states* forming an orthonormal basis for this two-dimensional complex vector space. Fundamentally, a qubit exists in a *continuum* of states between $|0\rangle$ and $|1\rangle$ until it is observed. Quantum mechanics tells us that we cannot examine a qubit to determine its quantum state, that is the value of $(\alpha, \beta)^T$, instead when we measure the qubit we get either result 0 with probability $|\alpha|^2$ or result 1 with probability $|\beta|^2$. For example, given a qubit in the state

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle,$$

the measurement returns result 0 half of the time, i.e. probability $|1/\sqrt{2}|^2$, and result 1 half of the time. Now, suppose we have two qubits. A two qubit system has four computational basis states denoted as $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. A pair of qubits exists in a superposition of these four states:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

where the measurement result $x \in \{0, 1\}^2$ occurs with probability $|\alpha_x|^2$ (and $\sum_{x \in \{0, 1\}^2} |\alpha_x|^2 = 1$). Here, we call the complex coefficient α_x an *amplitude* associating with each computational basis state. More generally, we consider a system of n qubits. The computational basis states of this system are of the form $|x_1 x_2 \dots x_n\rangle$, and a quantum state of such a system is specified by 2^n amplitudes. For $n = 500$, 2^n is already bigger than the estimated number of atoms in the observable Universe!

Classical computer circuits consist of *wires* that are used to carry information around the circuit and *logic gates* that perform manipulations of the information, converting it from one form to another. Analogous to a classical computer, a quantum computer is built from a *quantum circuit* consisting of wires and elementary *quantum gates* to carry around and manipulate the quantum information. Mathematically, quantum gates on a single qubit can be described by two by two complex matrices. We denote the a single qubit quantum gate as $U \in \mathbb{C}^{2 \times 2}$. One can think of a quantum gate acting as a matrix–vector multiplication. Application of this quantum gate to a qubit in quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ results into another quantum state $|\psi'\rangle = U|\psi\rangle = \alpha'|0\rangle + \beta'|1\rangle$ and the normalization condition $|\alpha'|^2 + |\beta'|^2 = 1$ must be satisfied. Therefore, the condition on the quantum gate is the matrix representing it must be *unitary*, i.e. $U^\dagger U = I$ where U^\dagger is the *adjoint* or *Hermitian transpose* of U . One of most useful gates is the *Hadamard* gate,

$$H \triangleq \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

We have:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle,$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle.$$

This gate is also called the *square-root of NOT* gate that turns $|0\rangle$ and $|1\rangle$ into their *halfways*. Algebra shows that $H^2 = I$, thus applying H twice to a quantum state does nothing to it. In general, for a quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, application of the Hadamard gate results into:

$$H|\psi\rangle = \alpha \frac{|0\rangle + |1\rangle}{\sqrt{2}} + \beta \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

In general, multiple qubit gates and quantum circuit can be modeled by unitary matrices. It is important to note that unitary quantum gates are *always* invertible, because the inverse of a unitary matrix is also a unitary matrix, and thus a quantum gate can always be inverted by another quantum gate. In contrast, in classical computing, given the output $A \oplus B$ from an XOR gate, it is not possible to determine what the inputs A and B were; so there is an irretrievable *loss of information* in the classical case. Essentially, the power of quantum mechanics for computation can be harnessed via reversible or invertible computation.

Quantum parallelism is a fundamental feature of many quantum algorithms. Heuristically, quantum parallelism allows quantum computers to evaluate a function $f(x)$ for many different values of x simultaneously. For example, suppose $f : \{0, 1\} \rightarrow \{0, 1\}$ is a function with a one-bit domain and range. To compute this function, we consider a two qubit quantum computer which starts in the state $|x, y\rangle$. With an appropriate sequence of logic gates, it is possible to transform this state into $|x, y \oplus f(x)\rangle$, where \oplus denotes the addition modulo 2, the first register is called the “data” register while the second one is called the “target”. We denote this transformation as a unitary U_f . If the target register is prepared as $y = 0$, then the final state of the second qubit is just the value $f(x)$, because $0 \oplus f(x)$ is $f(x)$ itself. If the data register is prepared as $x = (|0\rangle + |1\rangle)/\sqrt{2}$ which can be created with a Hadamard gate acting on $|0\rangle$, then $U_f|x, y\rangle$ results in the state:

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}},$$

in which different terms contain information about both $f(0)$ and $f(1)$; almost as if we have evaluated $f(x)$ for two values of x at the same time. In classical parallelism, multiple classical circuits, in which each is built to compute $f(x)$, are executed simultaneously. In quantum parallelism, a single quantum circuit is employed to evaluate the function for multiple values of x simultaneously, by exploiting the superposition nature of different states. This procedure

can be generalized to functions on an arbitrary number of bits, by using the *Hadamard transform* or *Walsh-Hadamard transform*. We consider n Hadamard gates acting in parallel on n qubits, denoted as $H^{\otimes n}$ where \otimes is read as tensor product. For instance, in case $n = 2$ with qubits initialized as $|0\rangle$, $H^{\otimes 2}$ gives

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) = \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}$$

as output. In general, $H^{\otimes n}$ acts on n qubits initialized in all $|0\rangle$ results into

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle.$$

Quantum parallel evaluation of a function with an n bit input x and 1 bit output, $f(x)$, can be performed by:

1. Prepare $n + 1$ qubit state $|0\rangle^{\otimes n}|0\rangle$,
2. Apply the Hadamard transform to the first n qubits, followed by the quantum circuit implementing U_f ,
3. The output is

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x, f(x)\rangle.$$

Quantum parallelism enables all possible values of the function f to be evaluated simultaneously, even though we evaluated f once. However, the measurement of the output state $\sum_{x \in \{0,1\}^n} |x, f(x)\rangle$ gives only $f(x)$ for a single value of x . Therefore, quantum computing requires more than quantum parallelism to be useful. Interested readers are recommended to read (Nielsen and Chuang, 2010) for further reference.

So far, in this introduction, we have warmed up for Quantum Fourier Transform (QFT)

with superposition, Hadamard gate and Hadamard transform. In the previous section 9.4.3 about Fast Fourier Transform (FFT) for group $(\mathbb{Z}/2\mathbb{Z})^k$, we have seen that the Hadamard transform plays the role of a Fourier transform of function defined on group $(\mathbb{Z}/2\mathbb{Z})^k$. Interestingly, QFT is also equivalent to the Hadamard transform.

As we already discussed, the Discrete Fourier Transform (DFT) is defined as transforming a set of N complex numbers $\{x_0, x_1, \dots, x_{N-1}\}$ into another set of complex numbers $\{y_0, y_1, \dots, y_{N-1}\}$ as follows:

$$y_k \triangleq \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k / N} x_j.$$

The Quantum Fourier Transform (QFT) on an orthonormal basis $|0\rangle, \dots, |N-1\rangle$ is defined to be a linear operator with the following action on the basis states,

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k / N} |k\rangle.$$

Equivalently, the action on an arbitrary state may be written as

$$\sum_{j=0}^{N-1} x_j |j\rangle \rightarrow \sum_{k=0}^{N-1} y_k |k\rangle,$$

where the amplitudes y_k are the DFT of the amplitudes x_j . The quantum gate acting on quantum state vectors is the following unitary matrix

$$F_N = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & w^3 & \cdots & w^{N-1} \\ 1 & w^2 & w^4 & w^6 & \cdots & w^{2(N-1)} \\ 1 & w^3 & w^6 & w^9 & \cdots & w^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & w^{N-1} & w^{2(N-1)} & w^{3(N-1)} & \cdots & w^{(N-1)(N-1)} \end{pmatrix},$$

where $w = e^{2\pi i/N}$ is the N -th root of unity. This transformation is a unitary transformation, and thus can be implemented as the dynamics for a quantum computer. It is possible to construct a manifestly quantum circuit computing the Fourier transform as follows.

In the following, we take $N = 2^n$, and the basis $|0\rangle, \dots, |2^n - 1\rangle$ is the computational basis for an n qubit quantum computer. We write the state $|j\rangle$ using the binary representation $j = j_1 j_2 \dots j_n$. Formally, $j = j_1 2^{n-1} + j_2 2^{n-2} + \dots + j_n 2^0$. We adopt the notation $0.j_\ell j_{\ell+1} \dots j_m$ to represent the binary fraction $j_\ell/2 + j_{\ell+1}/4 + \dots + j_m/2^{m-\ell+1}$. With some algebra, we can write the QFT in a useful form of *product representation* as follows:

$$|j\rangle \rightarrow \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i j k / 2^n} |k\rangle = \frac{1}{2^{n/2}} \bigotimes_{\ell=1}^n \left[\sum_{k_\ell=0}^1 e^{2\pi i j k_\ell 2^{-\ell}} |k_\ell\rangle \right] = \frac{1}{2^{n/2}} \bigotimes_{\ell=1}^n [|0\rangle + e^{2\pi i j 2^{-\ell}} |1\rangle],$$

or equivalently

$$|j_1, \dots, j_n\rangle \rightarrow \frac{(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)(|0\rangle + e^{2\pi i 0.j_n j_{n-1}}|1\rangle) \dots (|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n}|1\rangle)}{2^{n-2}}.$$

The action of the Quantum Fourier Transform can be expressed in a compact manner:

$$\text{QFT}(|j_1, \dots, j_n\rangle) = \frac{1}{\sqrt{N}} \left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle \right) \otimes \left(|0\rangle + e^{2\pi i 0.j_n j_{n-1}}|1\rangle \right) \otimes \dots \otimes \left(|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n}|1\rangle \right), \quad (9.8)$$

where \otimes denotes the tensor product. The product representation in Eq. 9.8 makes it simple to derive an efficient circuit for the QFT. The quantum gates used in the circuit are the Hadamard gate and the controlled phase gate R_k as follows:

$$H \triangleq \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{and} \quad R_k \triangleq \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i / 2^k} \end{pmatrix}.$$

The circuit composed of H gates and the controlled version of R_k is depicted in Figure 9.3.

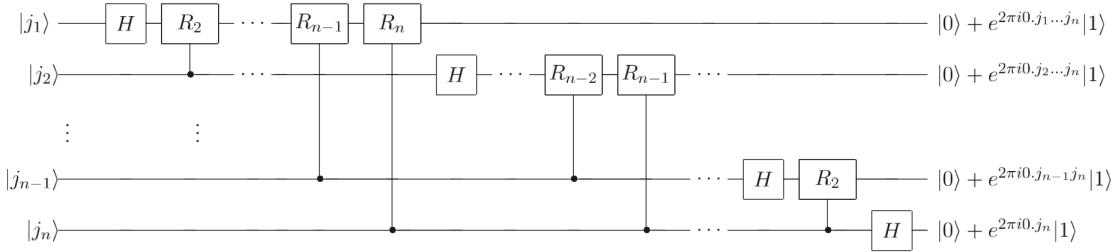


Figure 9.3: Efficient circuit derived from the product representation from Eq. 9.8 for the QFT. Swap gates at the end of the circuit which reverse the order of the qubits, and normalization factor of $1/\sqrt{2}$ of the outputs are not shown. At most $n/2$ swaps are needed at the end. This figure is taken from (Nielsen and Chuang, 2010).

Now, we analyze the number of gates in this circuit. We start by doing a Hadamard gate and $n - 1$ controlled phase gates (i.e. conditional rotations) on the first qubit – a total of n gates. The next one requires a Hadamard gate and $n - 2$ controlled phase gates, and each following term requires one fewer controlled phase gates. Summing up, we need $n(n + 1)/2$ gates. Therefore, this circuit gives us a $\Theta(n^2)$ algorithm for performing the QFT. In contrast, the best classical algorithm for computing DFT on 2^n elements is the FFT using $\Theta(n2^n)$ gates. That means we need exponentially more operations to compute Fourier transform on a classical computer than it does to implement the QFT on a quantum computer. It is possible to utilize the QFT to efficiently solve several problems that are believed to have no efficient solution on a classical computer. One of them is the *hidden subgroup problem* stated as “*Let f be a function from a finitely generated group G to a finite set X such that f is constant on the cosets of a subgroup H , and distinct on each coset. Given a quantum black box for performing the unitary transform $U|g\rangle|x\rangle = |g\rangle|x \oplus f(g)\rangle$, for $g \in G$ and $x \in X$, and \oplus is an appropriately chosen binary operation on X , find a generating set for H .*”. For readers who are interested in quantum computing and quantum information, (Nielsen and Chuang, 2010) is one of the most valuable resources for background of the subjects.

CHAPTER 10

APPENDIX: WAVELET THEORY

10.1 Chapter Introduction

Fourier and wavelet bases decompose signals over oscillatory waveforms that reveal many signal properties and signal regularity through the amplitude of coefficients, and their structures lead to fast computational algorithms and provide sparse representations. However, as opposed to Fourier bases, wavelets are well localized and few coefficients are needed to represent local transient structures. Historically, the Haar sequence (Haar, 1910) was first proposed by Alfréd Haar to construct a piecewise constant function

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \leq t < 1/2, \\ -1 & \text{if } 1/2 \leq t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

The dilations and translations of $\psi(t)$ generate an orthonormal basis of the space of square-integrable functions $L^2(\mathbb{R})$,

$$\left\{ \psi_{j,n}(t) = \frac{1}{\sqrt{2^j}} \psi\left(\frac{t - 2^j n}{2^j}\right) \right\}_{(j,n) \in \mathbb{Z}^2}.$$

Suppose we are given a signal f having a finite energy, i.e.

$$\|f\|^2 = \int_{-\infty}^{+\infty} |f(t)|^2 dt < +\infty,$$

and the inner product

$$\langle f, g \rangle = \int_{-\infty}^{+\infty} f(t)g^*(t)dt.$$

Any finite energy signal f can be represented as a linear combination of the wavelet orthonormal basis,

$$f = \sum_{j=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} \langle f, \psi_{j,n} \rangle \psi_{j,n},$$

where

$$\langle f, \psi_{j,n} \rangle = \int_{-\infty}^{+\infty} f(t) \psi_{j,n}(t) dt$$

are the wavelet inner-product coefficients. Orthogonal wavelets dilated by 2^j carry signal variations at the resolution 2^{-j} . The Haar sequence is now recognized as the first known and the simplest possible wavelet basis. The technical disadvantage of the Haar wavelet is that it is not continuous, and therefore not differentiable. However, large wavelet coefficients are located at sharp signal transitions, that can be an advantage for the analysis of discrete signals with sudden transitions. For image processing, wavelet orthonormal bases of images can be constructed from wavelet orthonormal bases of one-dimensional signals: three mother wavelets $\psi^1(x)$, $\psi^2(x)$, and $\psi^3(x)$ with $x = (x_1, x_2) \in \mathbb{R}^2$, are dilated by 2^j and translated by $2^j n$ with $n = (n_1, n_2) \in \mathbb{Z}^2$. Given an two-dimensional image or a finite energy function $f(x) = f(x_1, x_2)$, we define the set

$$\left\{ \psi_{j,n}^k(x) = \frac{1}{2^j} \psi^k \left(\frac{x - 2^j n}{2^j} \right) \right\}_{j \in \mathbb{Z}, n \in \mathbb{Z}^2, 1 \leq k \leq 3}$$

as an orthonormal basis of the space $L^2(\mathbb{R})$. The support of a wavelet $\psi_{j,n}^k$ is a square of width proportional to the scale 2^j (see Fig. 10.1). Wavelet coefficients for an image of N pixels can be calculated efficiently with a fast $O(N)$ algorithm (see (Mallat, 2008)).

Inspired by original ideas developed in computer vision to analyze images at several resolutions, (Mallat, 1989a) established the systematic theory for constructing orthonormal wavelet bases through the elaboration of multiresolution signal approximations or multiresolution analysis. A Multiresolution Analysis (MRA) of the Lebesgue space $L^2(\mathbb{R})$ constructs

a sequence of nested subspaces of functions

$$\{0\} \cdots \subset \mathbb{V}_1 \subset \mathbb{V}_0 \subset \mathbb{V}_{-1} \subset \cdots \subset \mathbb{V}_{-n} \subset \mathbb{V}_{-(n+1)} \subset \cdots \subset L^2(\mathbb{R}),$$

where \mathbb{V}_{j+1} is a smoother part of \mathbb{V}_j . The further we go from right to left, the longer the length scale over which typical functions in \mathbb{V}_j vary, thus, projecting a function to $\mathbb{V}_{j+1}, \mathbb{V}_{j+2}, \dots$ amounts to analyzing it at different levels of resolution. (Mallat, 1989b) defines MRA on \mathbb{R} directly in terms of dilations and translations by six axioms as in Def. 10.1.1. Properties of orthogonal wavelets and multiresolution approximations also brought to light the *fast wavelet transform* algorithm decomposing signals of size N with $O(N)$ operations (Mallat, 1989b).

Definition 10.1.1 (Multiresolutions). A sequence $\{\mathbb{V}_j\}_{j \in \mathbb{Z}}$ of closed subspaces of $L^2(\mathbb{R})$ is a multiresolution approximation if the following six properties are satisfied:

- $\forall (j, k) \in \mathbb{Z}^2, f(t) \in \mathbb{V}_j \Leftrightarrow f(t - 2^j k) \in \mathbb{V}_j,$
- $\forall j \in \mathbb{Z}, \mathbb{V}_{j+1} \subset \mathbb{V}_j,$
- $\forall j \in \mathbb{Z}, f(t) \in \mathbb{V}_j \Leftrightarrow f\left(\frac{t}{2}\right) \in \mathbb{V}_{j+1},$
- $\lim_{j \rightarrow +\infty} \mathbb{V}_j = \bigcap_{j=-\infty}^{+\infty} \mathbb{V}_j = \{0\},$
- $\lim_{j \rightarrow -\infty} \mathbb{V}_j = \text{Closure}\left(\bigcup_{j=-\infty}^{+\infty} \mathbb{V}_j\right) = L^2(\mathbb{R}),$

and there exists θ such that $\{\theta(t - n)\}_{n \in \mathbb{Z}}$ is a Riesz basis of \mathbb{V}_0 .

Remark. An orthonormal basis $\{\theta(t - n)\}_{n \in \mathbb{Z}}$ of \mathbb{V}_0 can be provided by $\theta(t) = \frac{\sin \pi t}{\pi t}$. All other properties of multiresolution approximation are easily verified.

In this thesis, chapter 6 discusses the graph wavelets obtained by Multiresolution Matrix Factorization and its consequential wavelet neural networks learning graphs. In the appendix, section 9.4 of chapter 9 briefly introduces the classifical Fourier transform, its generalized version on finite groups and the corresponding efficient algorithms. Extension of Fourier

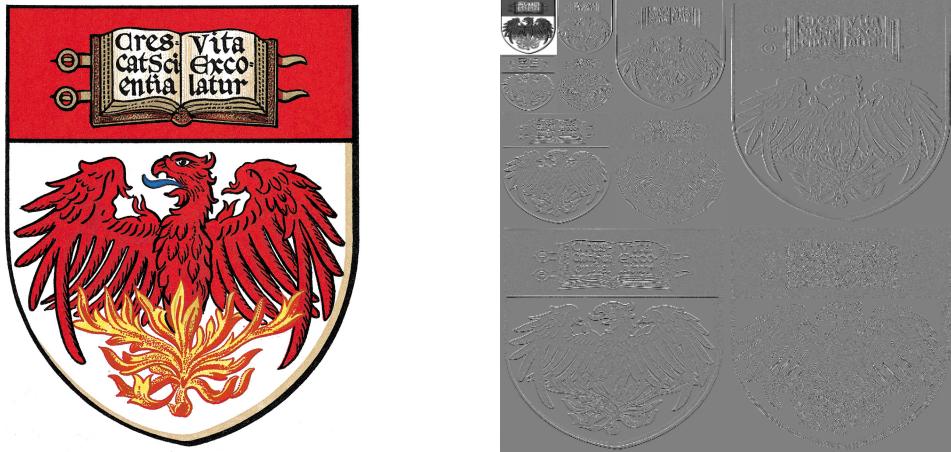


Figure 10.1: University of Chicago’s coat of arms and its Daubechies wavelet transform. The most commonly used set of discrete wavelet transforms was proposed by (Daubechies, 1988) in which the formulation is based on the use of recurrence relations to generate progressively finer discrete samplings of an implicit mother wavelet function; each resolution is twice of the previous scale. **Remark:** The coat of arms was resized to the size of 512×512 and uncolored as input to the wavelet transform.

transform on graphs have been mentioned in section 6.7 of chapter 6. Wavelet theory is such an important subject that deserves an its own introduction and this appendix serves the purpose. Section 10.2 introduces the classical wavelet transform, section 10.3 presents the spectral graph theory, and section 10.4 describes the spectral graph wavelets. It is important to note that wavelet theory and multiresolution analysis is a vast subject, and interested readers are recommended to read (Daubechies, 1992), (Mallat, 2008) and (Hammond et al., 2011).

10.2 Classical Wavelet Transform

This section gives an overview of the classical Continous Wavelet Transform (CWT) for $L^2(\mathbb{R})$, the set of square integrable real valued functions. In general, the CWT will be generated by the selection of a single “mother” wavelet, a continuously differentiable ψ , and then wavelets at different locations and spatial scales are formed by translating and scaling

the mother wavelet, i.e.

$$\psi_{s,a}(x) = \frac{1}{s} \psi\left(\frac{x-a}{s}\right),$$

where s must be a positive scale. Given a signal f , the wavelet coefficient at scale s and location a is given by the inner product of f with the wavelet $\psi_{s,a}$, i.e.

$$W_f(s, a) = \int_{-\infty}^{+\infty} \frac{1}{s} \psi^*\left(\frac{x-a}{s}\right) f(x) dx.$$

The inverse CWT is classically presented in the double-integral form. Assume you have a wavelet ψ with a Fourier transform that satisfies the admissibility condition:

$$C_\psi = \int_{-\infty}^{+\infty} \frac{|\hat{\psi}(w)|^2}{|w|} dw < \infty,$$

that also implies $\hat{\psi}(0) = \int \psi(x) dx = 0$, so ψ must have zero mean. For finite-energy function $f(x)$, we can define the inversion of the CWT as:

$$f(x) = \frac{1}{C_\psi} \int_0^{+\infty} \int_{-\infty}^{+\infty} W_f(s, a) \psi_{s,a}(x) \frac{da \cdot ds}{s}.$$

On graphs, the method of constructing the classical wavelet transform proceeds by producing the wavelets directly in the signal domain, through scaling and translation becomes problematic. Given a function $\psi(x)$ defined on the vertices of a weighted graph, i.e. x is a vertex of the graph, there is no interpretation of sx for a real scalar s , thus it is not trivial how to define $\psi(sx)$.

10.3 Spectral Graph Theory

One of the first ways to construct wavelets on graphs is spectral graph wavelet transform. Before discussing this, it is necessary to introduce about the theory behind, the spectral

graph theory which is a branch of mathematics to study properties of a graph in relationship to the eigenvalues, eigenvectors, and characteristic polynomial of matrices associated with the graph including its adjacency matrix and Laplacian matrices.

10.3.1 Graph Laplacian

A weighted undirected finite graph $G = (E, V, w)$ consists of a set of vertices V , a set of edges E , and a weight function $w : E \rightarrow \mathbb{R}^+$ which assigns a positive weight to each edge. The adjacency matrix $A = [a_{i,j}]$ is a matrix of size $|V| \times |V|$ with entries defined as

$$a_{i,j} = \begin{cases} w(e) & \text{if } e = (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Loops, i.e. edges that connect a single vertex to itself, imply the presence of nonzero diagonal entries in the adjacency matrix. Since the graph is weighted, the degree of each vertex v is defined as the sum of the weights of all the edges connecting to it, $d(v) \triangleq \sum_{v'} a_{v,v'}$. We define the diagonal matrix D that its diagonal elements are equal to the degrees. A graph signal can be viewed as a real valued function $f : V \rightarrow \mathbb{R}$.

The key importance of spectral graph theory is the graph Laplacian operator \mathcal{L} . The non-normalized Laplacian is defined as $\mathcal{L} \triangleq D - A$. For any graph signal $f \in \mathbb{R}^{|V|}$, \mathcal{L} satisfies

$$(\mathcal{L}f)(i) = \sum_{j:e=(i,j)\in E} w(e)(f(i) - f(j)).$$

The graph Laplacian can be defined for graphs arising from sampling points on a differentiable manifold. The regular mesh is a simple example of such a sampling process. In this case, the graph Laplacian corresponds to the standard stencil approximation of the continuous Laplacian. Consider the graph defined by taking vertices $v_{i,j}$ as points on a regular

two dimensional grid, with each point connected to its four neighbors with weight $1/(dx)^2$, where dx is the distance between adjacent grid points. Applying the graph Laplacian to a two-dimensional function $f_{i,j}$ defined on the grid yields

$$(\mathcal{L}f)_{i,j} = \frac{4f_{i,j} - f_{i+1,j} - f_{i-1,j} - f_{i,j+1} - f_{i,j-1}}{dx^2}$$

which is the standard 5-point stencil for approximating $-\nabla^2 f$.

On the another hand, the normalized graph Laplacian is defined as

$$\mathcal{L}^{norm} = D^{-1/2} \mathcal{L} D^{-1/2} = I - D^{-1/2} A D^{-1/2}.$$

Indeed, \mathcal{L} and \mathcal{L}^{norm} have different eigenvectors. Both operators may be used to define spectral graph wavelet transforms, but the resulting transforms will not be equivalent. In this chapter, we will use the non-normalized form of the Laplacian.

10.3.2 Graph Fourier Transform

On the real line, the inverse Fourier transform is defined as

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(w) e^{iwx} dw, \quad (10.1)$$

where the complex exponentials e^{iwx} are eigenfunctions of the one-dimensional Laplacian operator $\frac{d}{dx^2}$. Therefore, the inverse Fourier transform can be seen as the expansion of f in terms of the eigenfunctions of the Laplacian operator. The graph Fourier transform is indeed defined analogously.

Since the graph Laplacian \mathcal{L} is a real symmetric matrix, it has a complete set of orthonormal

eigenvectors $\{u_\ell\}_{\ell=0,\dots,|V|-1}$ with the associated real non-negative eigenvalues $\{\lambda_\ell\}_{\ell=0,\dots,|V|-1}$. The multiplicity of the zero eigenvalue is equal to the number of connected components of the graph. Assume that the graph is connected, i.e. as a single connected component, we can order the eigenvalues such that $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_{|V|-1}$. For any graph signal $f \in \mathbb{R}^{|V|}$, the graph Fourier transform is defined by

$$\hat{f}(\ell) \triangleq \langle u_\ell, f \rangle = \sum_{v=1}^{|V|} u_\ell^*(v) f(v),$$

and the inverse transform reads as

$$f(v) \triangleq \sum_{\ell=0}^{|V|-1} \hat{f}(\ell) u_\ell(v).$$

10.4 Spectral Graph Wavelet Transform

10.4.1 Construction

The spectral graph wavelet transform (SGWT) will be determined by the choice of a kernel function $\kappa : \mathbb{R}^+ \rightarrow \mathbb{R}^+$, that should behave as a band-pass filter, i.e. satisfying $\kappa(0) = 0$ and $\lim_{x \rightarrow +\infty} \kappa(x) = 0$. In particular, the wavelet operator $T_\kappa = \kappa(\mathcal{L})$ acts on a graph signal f by modulating each Fourier mode as $\widehat{T_\kappa f}(\ell) = \kappa(\lambda_\ell) \hat{f}(\ell)$. Employing the inverse Fourier transform yields

$$(T_\kappa f)(v) = \sum_{\ell=0}^{|V|-1} \kappa(\lambda_\ell) \hat{f}(\ell) u_\ell(v).$$

The wavelet operators at scale s is then defined by $T_\kappa^s = \kappa(s\mathcal{L})$. The spectral graph wavelets are realized via localizing these wavelet operators by applying them to the impulse on a

single vertex, i.e. $\psi_{s,a} = T_\kappa^s \delta_a$. Explicitly, we have

$$\psi_{s,a}(v) = \sum_{\ell=0}^{|V|-1} \kappa(s\lambda_\ell) u_\ell^*(a) u_\ell(v). \quad (10.2)$$

The wavelet coefficients of a graph signal f are obtained by taking the inner product with these wavelets, as $W_f(s, a) = \langle \psi_{s,a}, f \rangle$. Because $\{u_\ell\}$ are orthonormal, the wavelet coefficients can also be obtained directly from the wavelet operators, as

$$W_f(s, a) = (T_\kappa^s f)(a) = \sum_{\ell=0}^{|V|-1} \kappa(s\lambda_\ell) \hat{f}(\ell) u_\ell(a). \quad (10.3)$$

10.4.2 Main result

In general, for a particular transform to be useful for signal processing, it must be possible to reconstruct from a given set of transform coefficients (see Lemma 10.4.1).

Lemma 10.4.1. *If the SGWT kernel κ satisfies the admissibility condition*

$$\int_0^{+\infty} \frac{\kappa^2(x)}{x} dx = C_\kappa < +\infty,$$

and $\kappa(0) = 0$, then for all $v \in V$,

$$\frac{1}{C_\kappa} \sum_{a=1}^{|V|} \int_0^{+\infty} W_f(s, a) \psi_{s,a}(v) \frac{ds}{s} = \bar{f}(v), \quad (10.4)$$

where $\bar{f} = f - \langle u_0, f \rangle u_0$. In particular, the complete reconstruction is then given by $f = \bar{f} + \hat{f}(0)u_0$.

Proof. Given the expression of $\psi_{s,a}$ and $W_f(s, a)$ in Eq. 10.2 and Eq. 10.3, the left hand side

of 10.4 becomes

$$\frac{1}{C_\kappa} \int_0^{+\infty} \frac{1}{s} \sum_a \left(\sum_\ell \kappa(s\lambda_\ell) u_\ell(a) \hat{f}(\ell) \sum_{\ell'} \kappa(s\lambda_{\ell'}) u_{\ell'}^*(a) u_{\ell'}(v) \right) ds,$$

with rearranging terms, that also becomes

$$\frac{1}{C_\kappa} \int_0^{+\infty} \frac{1}{s} \left(\sum_{\ell, \ell'} \kappa(s\lambda_{\ell'}) \kappa(s\lambda_\ell) \hat{f}(\ell) u_{\ell'}(v) \sum_a u_{\ell'}^*(a) u_\ell(a) \right) ds.$$

The orthogonality of u_ℓ implies $\sum_a u_{\ell'}^*(a) u_\ell(a) = \delta_{\ell, \ell'}$. Thus, the left hand side of 10.4 is equal to

$$\frac{1}{C_\kappa} \int_0^{+\infty} \frac{1}{s} \left(\sum_{\ell, \ell'} \kappa(s\lambda_{\ell'}) \kappa(s\lambda_\ell) \hat{f}(\ell) u_{\ell'}(v) \delta_{\ell, \ell'} \right) ds,$$

that is also equal to

$$\frac{1}{C_\kappa} \sum_\ell \left(\int_0^{+\infty} \frac{\kappa^2(s\lambda_\ell)}{s} ds \right) \hat{f}(\ell) u_\ell(v). \quad (10.5)$$

Because κ satisfies the admissibility condition, i.e.

$$\int_0^{+\infty} \frac{\kappa^2(x)}{x} dx = C_\kappa < +\infty,$$

with the variable substitution $x = s\lambda_\ell$, we have:

$$C_\kappa = \int_0^{+\infty} \frac{\kappa^2(s\lambda_\ell)}{s\lambda_\ell} d(s\lambda_\ell) = \int_0^{+\infty} \frac{\kappa^2(s\lambda_\ell)}{s} d(s),$$

that is independent of ℓ , except for when $\lambda_\ell = 0$ at $\ell = 0$ and the integral is zero. The expression 10.5 becomes

$$\sum_{\ell=1}^{|V|-1} \hat{f}(\ell) u_\ell(v) = \sum_{\ell=0}^{|V|-1} \hat{f}(\ell) u_\ell(v) - \hat{f}(0) u_0(v) = f(v) - \langle u_0, f \rangle u_0(v) = \bar{f}(v).$$

Therefore, we proved our desired result. \square

Wavelets, in general, provide simultaneous localization in both frequency and time (or space). It can be shown that if the kernel κ is localized in the spectral domain, then the associated spectral graph wavelets will be all localized in frequency. Interested readers are recommended to read (Hammond et al., 2011) for further reference in the field.

10.5 Diffusion wavelets

10.5.1 *Diffusion operators*

In section 3.2.8, we have discussed the diffusion operators and kernels in the context of graphs. Diffusion wavelets proposed by (Coifman and Maggioni, 2006) are a fast multiresolution framework for the analysis of functions on discrete or discretized continuous structures such as graphs and manifolds, in which diffusion plays a role as a smoothing and scaling tool to enable coarse graining and multiresolution analysis. Diffusion wavelets have been used extensively in machine learning, for example: value function approximation in reinforcement learning (Mahadevan and Maggioni, 2005) and multiscale analysis of Markov decision processes (Maggioni and Mahadevan, 2006). This section serves as an introduction to this famous multiresolution method.

Given a diffusion operator T on a manifold or a graph, with large powers of low rank, (Coifman and Maggioni, 2006) presents a general multiresolution construction for efficiently computing, representing and compressing T^t that allows a direct multiscale computation, to high precision, of functions of the operator. For many examples arising in physics, the powers of the operator T decrease in rank, thus suggesting the compression of the function space upon which each power acts. The scheme of compression is described as follows: apply T to a space of test functions at the finest scale, compress the range via a local orthonormalization procedure, represent T in the compressed range and compute T^2 on this

range, compress its range and orthonormalize, and so on. At resolution j , we obtain a compressed representations of T^{2^j} , acting on a family of scaling functions spanning the range of $T^{1+2+2^2+\dots+2^{j-1}}$, for which we have a (compressed) orthonormal basis, and then we apply T^{2^j} , locally orthonormalize and compress the result, thus getting the next coarser subspace.

One application of this scheme is a generalization of the *Fast Multipole Method* (FMM) (Greengard and Rokhlin, 1987) for the efficient computation of the product of the inverse Laplacian $(I - T)^{-1}$ with an arbitrary vector f (Beylkin et al., 1991). A *Neumann series* is a mathematical series that generalizes the geometric series, of the form $\sum_{k=0}^{\infty} T^k$ where T is an operator and $T^k \triangleq T^{k-1} \circ T$ is its k times repeated application. It has been known that if the Neumann series converges in the operator norm, then $I - T$ is invertible and its inverse is the series: $(I - T)^{-1} = \sum_{k=0}^{+\infty} T^k$. Thus,

$$(I - T)^{-1} f = \left(\sum_{k=0}^{+\infty} T^k \right) f.$$

By letting $S_K = \sum_{k=0}^{2^K} T^k$, we can write

$$S_{K+1} = S_K + T^{2^K} S_K = \prod_{k=0}^K (I + T^{2^k}),$$

and furthermore, we get

$$(I - T)^{-1} f \approx \prod_{k=0}^K (I + T^{2^k}) f,$$

for large K . Since the powers T^{2^k} have been compressed and can be applied efficiently to f , we can express $(I - T)^{-1}$ in compressed form and efficiently apply it to any function f . This is quite similar to our Multiresolution Matrix Factorization (MMF) described in chapter 6 in the sense that MMF factorizes the matrix into a product of several sparse matrices (until a certain resolution) that allows faster matrix-vector multiplication.

10.5.2 Algorithm & Discussion

We present the algorithm to construct the scaling basis functions and the wavelet basis functions along with the representations of the diffusion operator T at these resolutions (Coifman and Maggioni, 2006) as follows. For the notation,

- Φ_i and Ψ_j denote the scaling basis functions at scale i and the wavelet basis functions at scale j , respectively.
- $[\Phi_j]_{\Phi_i}$ denotes the matrix representation of the scaling basis Φ_j represented with respect to the basis Φ_i .
- $[T]_{\Phi_j}^{\Phi_i}$ denotes the matrix representing the operator T , where the row space of T (i.e. the domain) is represented with respect to the basis Φ_i , and the column space of T (i.e. the range) is represented with respect to the basis Φ_j .
- SpQR denotes the function for sparse QR factorization with the template $Q, R \leftarrow \text{SpQR}(A, \epsilon)$, in which
 - **Input:** A is a sparse $n \times n$ matrix, and $\epsilon = 10^{-6}$ is the precision.
 - **Output:** $Q \in \mathbb{R}^{n \times m}$ orthogonal and $R \in \mathbb{R}^{m \times n}$ upper triangular such that $A \approx_{\epsilon} QR$. These matrices are possibly sparse.

The algorithm is described as in Algorithm 19. Fig. 10.2 illustrates the algorithm.

In a certain way, diffusion wavelets inspired our Multiresolution Matrix Factorization (MMF) (see chapter 6), but MMF is more general in the sense that it need not to lean on the assumption of low rank of large powers (i.e. T^k has a low rank if k is large) and it is applicable for a wide range of hierarchical matrices in general. Furthermore, MMF coarsens the set of active columns (or nodes of a graph if the matrix is the graph Laplacian) by aggressively dropping columns, that is the unique feature of multiresolution construction of Multiresolution Matrix Factorization.

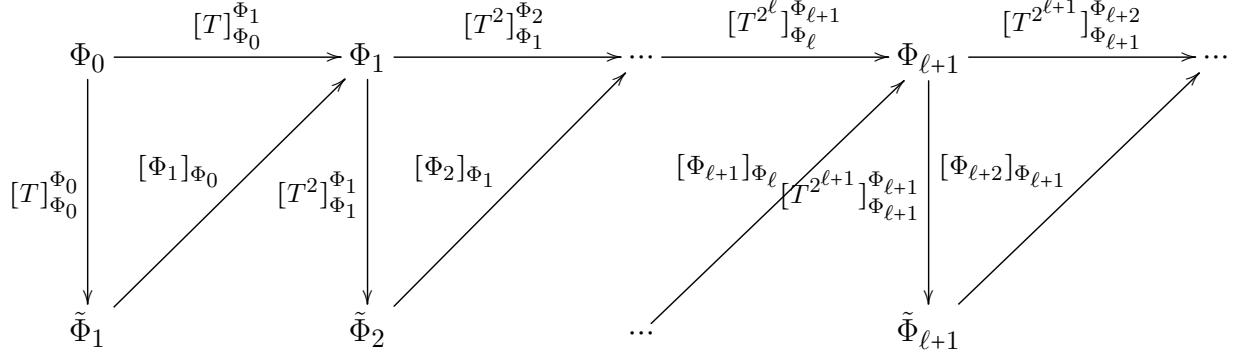


Figure 10.2: Diagram for downsampling, orthogonalization and operator compression (Coifman and Maggioni, 2006).

Algorithm 19 Pseudo-code for construction of a Diffusion Wavelet Tree (Coifman and Maggioni, 2006).

- 1: **Input:** T is the matrix representation of the diffusion operator, ϵ is the precision of the QR decomposition (e.g., 10^{-6}), and L is the maximum number of resolution levels.
 - 2: **Output:** $\{\Phi_\ell\}$ and $\{\Psi_\ell\}$ that are the sets of scaling and wavelet basis functions indexed by resolution ℓ , respectively.
 - 3: **for** $\ell = 0 \rightarrow L - 1$ **do**
 - 4: $[\Phi_{\ell+1}]_{\Phi_\ell}, [T^{2^\ell}]_{\Phi_\ell}^{\Phi_{\ell+1}} \leftarrow \text{SpQR}\left([T^{2^\ell}]_{\Phi_j}^{\Phi_j}, \epsilon\right)$
 - 5: $[T^{2^{\ell+1}}]_{\Phi_{\ell+1}}^{\Phi_{\ell+1}} \leftarrow \left([T^{2^\ell}]_{\Phi_\ell}^{\Phi_{\ell+1}} [\Phi_{\ell+1}]_{\Phi_\ell}\right)^2$
 - 6: $[\Psi_\ell]_{\Phi_\ell} \leftarrow \text{SpQR}\left(I_{\langle \Phi_\ell \rangle} - [\Phi_{\ell+1}]_{\Phi_\ell} [\Phi_{\ell+1}]_{\Phi_\ell}^*\right)$
 - 7: **end for**
-