Inductive biases for graph neural networks

García Satorras, V.

Publication date
2024
Document Version
Final published version

Inductive Biases for Graph Neural Networks

Inductive Biases for Graph Neural Networks

Victor Garcia Satorras

Victor Garcia Satorras

Inductive Biases for Graph Neural Networks

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag 15 mei 2024, te 10.00 uur

door Victor García Satorras
geboren te Barcelona

Promotiecommissie

Promotor:	prof. dr. M. Welling	Universiteit van Amsterdam
Copromotor:	dr. H.C. van Hoof	Universiteit van Amsterdam
Overige leden:	prof. dr. W. Boomsma	University of Copenhagen
	dr. I.A. Titov	Universiteit van Amsterdam
	prof. dr. C. Monz	Universiteit van Amsterdam
	dr. D. Thanou	Ecole polytechnique fédérale de Lausanne
	dr. E. Gavves	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

ABSTRACT

Graph structured representations are a powerful inductive bias applicable across a wide spectrum of systems in nature, ranging from atom interactions in molecular systems to complex human interactions such as social networks. Part of the success of Graph Neural Networks (GNNs) can be attributed to their broad applicability in capturing these complex interactions. This thesis aims to extend the capabilities of GNNs by incorporating additional physics-based inductive biases.

The thesis begins by enriching GNN architectures with traditional graphical inference methods to craft hybrid models. These models leverage the prior knowledge inherent in conventional graphical models along with the adaptive inference from data-driven learning. The resulting algorithms outperform the individual approaches when run in isolation.

We then implement inductive biases as a symmetry constraint by creating E(n) Equivariant Graph Neural Networks (EGNNs), which improve upon standard GNNs through the inclusion of the Euclidean symmetry. This improves generalization for data within n-dimensional Euclidean spaces, a characteristic particularly relevant in molecular data. Subsequently, we demonstrate the benefits of EGNNs in various applications in the domain of deep learning for molecular modelling.

The concluding part of this work is the incorporation of euclidean symmetries into generative models by building upon the proposed EGNNs. The presented generative model significantly outperforms previous 3D molecular generative models, showing potential to be disruptive in the future of molecular modeling.

SAMENVATTING

Representaties die gestructureerd zijn als grafen zijn een belangrijke inductieve aanname die breed toepasbaar is op natuurlijke systemen, van atomische interacties in moleculaire systemen tot complexe menselijke interacties in sociale netwerken. Een gedeelte van het succes van Graph Neural Networks (GNNs) kan worden toegeschreven aan hun toepasbaarheid in het modelleren van deze brede complexe interacties. Deze dissertatie verbetert de capaciteiten van GNNs door aanvullende inductieve aannames toe te voegen die op natuurkunde gebaseerd zijn.

De dissertatie begint met het verrijken van GNN architecturen door gebruik te maken van traditionele *graphical inference* methoden om hybride modellen te vormen. Deze modellen combineren de a priori kennis van conventionele *graphical models* met de adaptieve inferentie uit data. Deze gecombineerde algorithmen presenteren beter dan de componenten waaruit ze bestaan.

Vervolgens implementeren we inductieve aannames als symmetrie randvoorwarde door E(n) Equivariant Graph Neural Networks (EGNNs) te maken. Deze verbeteren standaard GNNs door Euclidische symmetrieën toe te voegen. Dit verbetert generalisatie voor data in n-dimensionale Euclidische ruimten, een eigenschap die in het bijzonder relevant is voor moleculaire data. Vervolgens demonstreren we de voordelen van EGNNs met verschillende toepassingen in het domein van *deep learning* voor moleculair modelleren.

Het concluderende deel van dit werk verenigt Euclidische symmetrie met generatieve modellen door EGNNs verder te ontwikkelen. De gepresenteerde generatieve modellen werken beter dan vorige 3D moleculair generatieve modellen, en dragen daarmee bij aan de toekomst van moleculair modelleren.

MAIN PUBLICATIONS

This thesis is based on the following publications:

- Victor Garcia Satorras, Zeynep Akata, and Max Welling. "Combining generative and discriminative models for hybrid inference." *Advances in Neural Information Processing Systems 32*. Short Oral, 2019.

- Victor Garcia Satorras, and Max Welling. "Neural enhanced belief propagation on factor graphs." *International Conference on Artificial Intelligence and Statistics.* Oral, 2021.

- Victor Garcia Satorras , Emiel Hoogeboom, and Max Welling. "E(n) Equivariant Graph Neural Networks." *International Conference on Machine Learning*. Poster, 2021.

- Victor Garcia Satorras*, Emiel Hoogeboom*, Fabian Fuchs, Ingmar Posner and Max Welling. "E(n) Equivariant Normalizing Flows." *Advances in Neural Information Processing Systems 34*. Oral, 2021.

- Emiel Hoogeboom*, Victor Garcia Satorras*, Clement Vignac* and Max Welling. "Equivariant Diffusion for Molecule Generation in 3D." *International Conference on Machine Learning*. Oral, 2022.

For all the above publications Victor Garcia Satorras contributed to the majority of the ideas, coding experiments and writing the paper. The final two papers were done in equal contribution with authors denoted by an asterisk (*) who also contributed to the main idea, experiments and writing the paper. Detailed acknowledgements for these equal contributions are provided in their respective chapters. Throughout the entire research Max Welling offered guidance, insightful feedback, and supervision. The remaining authors all contributed with valuable suggestions and discussions.

The author further contributed to the following publications during the course of his PhD:

- Victor Garcia Satorras and Joan Bruna. "Few-shot learning with graph neural networks." *International Conference on Learning Representations.* 2018.

- Emiel Hoogeboom, Victor Garcia Satorras, Jakub Tomczak and Max Welling. "The convolution exponential and generalized sylvester flows." *Advances in Neural Information Processing Systems 33*. 2020.

- Fiorella Wever, T. Anderson Keller, Laura Symul, Victor Garcia. "As easy as APC: overcoming missing data and class imbalance in time series with self-supervised learning." *arXiv preprint arXiv:2106.15577*. 2021.

- Daniel Zügner, Francois-Xavier Aubet, Victor Garcia Satorras, Tim Januschowski, Stephan Günnemann and Jan Gasthaus. "A study of joint graph inference and forecasting." *Time Series Workshop ICML 2021*.

- Victor Garcia Satorras, Syama Sundar Rangapuram, and Tim Januschowski. "Multivariate Time Series Forecasting with Latent Graph Inference." *arXiv preprint arXiv:2203.03423*. 2022.

- Ilia Igashov, Hannes Stärk, Clement Vignac, Victor Garcia Satorras, Pascal Frossard, Max Welling, Michael Bronstein, Bruno Correia. "Equivariant 3d-conditional diffusion models for molecular linker design." *arXiv preprint arXiv:2210.05274.* 2022

ACKNOWLEDGEMENTS

As I finish writing the final sections of this thesis, it is not the technical content that comes to mind but the people, the city of Amsterdam, and all the experiences that have shaped my PhD journey during these quick and intense four years.

First and foremost, I wish to express my heartfelt gratitude to my supervisor, Max Welling. Your enthusiasm and passion for research have been inspirational and contagious. From you, I have learned the importance of being driven by curiosity and ideas, and that true impact emerges from a sincere study and understanding of the field. My thanks also extend to my co-promotors, Zeynep Akata and Herke van Hoof. Zeynep, who was my co-supervisor at the start of the PhD, welcomed me into the lab and always offered support. Herke, whose prompt responses and professionalism have been very valuable.

Moving to the core of this journey - the DeltaLab office is one of the first things that pop into my mind. Special mention goes to my friends and colleagues Andy and Emiel. You both taught me that fun and work are not mutually exclusive but can actually go together. In fact, it was in the moments of fun where the best ideas were born. To Andy, I am grateful for all the spontaneous adventures we had around Amsterdam and your unique sense of humor, without you, my experience of the city would not have been the same. To Emiel, I am thankful not only for having you as a friend but also for being the best collaborator I had the luck to work with. I want to extend my thanks to all Delta Lab members, Jorn P., Sindy L., Sadaf G., Elise v. d. P., Ivan S., Artem M., and Wendy S., with whom we shared many afterworks, retreats, a ski trip but more importantly, the unforgettable trips to Renningen. I also want to extend this thanks to Amlab, to Marco F., Thomas K., Teddy, Benjamin M., Putri v. d. L., David R., Wouter K., Ana L., Bas V., Tineke B., Tim B., Sara M., Babak E., Heiko Z., Fiona L., Rianne v. d. B., Patrick F., Erik B., Felice A., and to everyone else in Amlab, thanks for the insightful discussions, coffee chats, and drinks at Oerknal. Special thanks to Patrick F. for his invaluable mathematical proofs and to Felice A. for all the administrative work.

The students I supervised deserve a special mention: Haitam, Fiorella, and Vasileios. I am truly proud of your achievements and the passion you put into your theses. Your dedication and perseverance have taught me a lot in return. Similarly, I am grateful to my supervisors during my internship in Berlin, Tim J. and Syama R., for their guidance.

Shifting countries and moving to Amsterdam was an intimidating step, but I found a *home* in the form of friends. Sara, Oriol, Nacho, Maria, Hugo, and Noe, I am very grateful for these shared years and experiences. Special thanks to Sara, who has been a big source of support over the last five years; thank you for always lending an ear to my endless stories, bringing a fresh perspective and invaluable advice.

Along the journey, I made new friends in Amsterdam - Eva, Pau, Toufic, Charlotte, Juan, Callum, Jose, Giulia and a frequent visitor Marc d. M. Your support, friendship and padel matches made me like the city even more. A special thanks to Giulia for the sharp feedback when proofreading the thesis.

I owe a huge gratitude to all my friends in Barcelona who have been my roots and a constant source of advice and support. In the context of this thesis I want to explicitly mention Miriam B, Xavi P. and Amaia S., for starting the journey in machine learning together at the UPC, and to Xavi G. for giving us the opportunity to present our work in our home town university every year.

Lastly, the foundation of who I am today has been deeply influenced by my family. My deepest gratitude is for my parents and my brother. Thank you for always supporting me through the highs and lows in every new life adventure I take.

CONTENTS

1

INTRODUCTION

1.1 OVERVIEW

Inductive biases are essential in Machine Learning for enabling algorithms to generalize from a limited set of training data to novel unseen examples. Such biases are critical in guiding the learning process towards viable solutions by leveraging inherent structure or symmetries in the data. For instance, Convolutional Neural Networks (CNNs) (LeCun et al., 1998) exhibit translation equivariance, allowing them to recognize patterns regardless of their position in the visual field. Additionally, neural networks commonly incorporate an inherent smoothness bias, which assumes that small variations in input lead to small variations in the output, reflecting an expectation of continuity in the data they model. These biases are key in the design of neural networks and significantly influence their learning and generalization capabilities.

Structuring data as a graph (a set of interconnected nodes) is an inductive bias observed in nature across various scales. At the granular level of chemistry, molecules can be modeled as a set of atoms interacting with each other. In biology, cellular networks emerge from the intricate connections between cells. On a larger scale, social networks arising from human interactions can similarly be represented as graphs. Even larger chaotic systems, such as planetary systems, or complex abstract concepts can be structured as graphs, such as knowledge graphs.

Graph Neural Networks (GNNs) (Bruna et al., 2013) leverage the inductive biases present in graph-structured data. Part of their success can be attributed to their broad applicability in capturing complex interactions in graph data. However, similar to other deep learning architectures, GNNs may require considerable amounts of training data to achieve robust generalization to new observations.

In many real-world systems, we can achieve better generalization by leveraging inductive biases related to the dynamics of a system. The inductive biases considered in this thesis fall into two categories: i) Physics-based equations that approximate the dynamics of real-world systems. ii) Symmetry constraints that must be strictly satisfied, such as the invariance of molecular properties to their orientation.

In the first part of the thesis, we focus on integrating physics-based inductive biases with data-driven models. These physics biases are structured within graphical models, providing a structural representation of the system. Example applications include the motion equations in a tracking system or the differential equations used to model chaotic phenomena relevant to applications such as weather forecasting. While graphical models defined from physics approximations provide robust generalization to new data points, they may not capture all the intricate details present in real-world data. Our goal in the first part of the thesis is to leverage the best from both worlds: the high generalizability of graphical models and the high flexibility of GNNs. For this purpose, we explore hybrid message-passing models that combine both graphical inference and learned inference from data.

In the subsequent parts of the thesis, we focus on leveraging symmetries that must be strictly satisfied in certain settings, such as invariance or equivariance to certain Euclidean transformations. This principle is particularly relevant for data that lies in a 3D space, such as molecular data, offering the potential for significant real-world impact. We start by building a graph neural network equivariant to Euclidean symmetries and demonstrate its efficacy in a series of discriminative tasks such as molecular property prediction. We then build upon this network to design equivariant generative models which significantly outperform previous methods for molecule generation in 3D, resulting in a promising approach for molecular modelling and drug discovery.

In summary, this thesis explores the incorporation of inductive biases into graph neural networks to design models that are better grounded in reality with the potential to better address real world applications as for example deep learning for molecular modelling.

1.2 SCOPE AND RESEARCH QUESTIONS

Research Question 1: *How can we benefit from both the expressivity of GNNs and the generalization and data efficiency of graphical models?*

A graphical model is a structured representation of the data generation process, however, the equations that define a graphical model are often only a poor approximation of the true data generation process. The subtleties of the generative

process are however captured in the data, and if data is available, we can learn to infer the missing subtleties from it, but pure learned inference can be data hungry and lack generalization.

In Chapters 3 and 4, we propose a hybrid message-passing framework that combines both graphical inference and learned inference, which we structure as a Graph Neural Network. By using cross-validation, we can automatically balance the amount of work performed by graphical inference versus learned inference. Specifically, in Chapter 3, we propose a hybrid framework for a Gaussian Hidden Markov Model, the type of graphical model used in Kalman Filters. We demonstrate through a variety of experiments that we can estimate trajectories much more accurately than either learned or graphical inference run in isolation. In Chapter 4, we extend the proposed framework to enhance Belief Propagation, a standardized algorithm used to reason over graphical models.

Research Question 2: *How can we build effective and yet fast graph neural networks that are equivariant to Euclidean transformations?*

In Chapter 5, we introduce E(n) Equivariant Graph Neural Networks (EGNNs). In contrast to previous existing E(n) equivariant methods, EGNNs do not require computationally expensive higher-order representations (spherical harmonics) allowing for significantly faster inference speeds, while still achieving competitive or better performance. Moreover, while existing methods are limited to equivariance on 3 dimensional spaces, the proposed model can scale to higher-dimensional spaces. We demonstrate the effectiveness of the EGNN on dynamical systems modelling, representation learning in graph autoencoders and predicting molecular properties. Due to its benefits, the EGNN has been adopted in various subsequent works in molecular modelling, and in our case, we further used it to design equivariant generative models to answer our next Research Question.

Research Question 3: *How can we construct E(3) equivariant generative models and what advancements do they offer in the generation of molecular structures?*

Generative models have shown significant breakthroughs in the domains of image and text generation, and they are promising candidates to impact the field of drug discovery via molecular modeling. Molecules inherently exist within a 3-dimensional space, subjecting them to Euclidean symmetries, such as rotations and translations. Leveraging these symmetries can enhance the generalization capabilities of generative models for molecular structures.

In Chapters 6 and 7, we build upon the EGNN framework from the previous chapter to develop two novel equivariant generative models: the E(n) Equivariant Normalizing Flows (E-NFs) and the E(3) Equivariant Diffusion Model (EDM).

The EDM, in particular, marks a significant improvement in the performance of molecular generative models. Additionally, we experiment with EDM to condition the generative process on targeted properties, aiming to produce molecules with specific characteristics.

1.3 SUBSEQUENT IMPACT

In this section, we discuss some of the subsequent impacts on the literature derived from the research presented in this thesis.

E(n) Equivariant Graph Neural Networks (discussed in Chapter 5) have been used in a wide variety of applications within the field of molecular modelling. In addition to parameterizing the molecular generative models (Satorras et al., 2021a; Hoogeboom et al., 2022) we introduce in Chapters 6 and 7, the EGNN has also been used in the literature to model binding interactions between ligands and molecules/proteins, a key problem in drug discovery. Some works use the EGNN to construct conditional generative models of ligands (Igashov et al., 2022; Schneuing et al., 2022; Rozenberg and Freedman, 2022), while others use or adapt the EGNN in a discriminative manner to predict structure coordinates in binding interactions (Stärk et al., 2022; Masters et al., 2022; Dhakal et al., 2023; Sestak et al., 2023; Zhang et al., 2023; Guan et al., 2023).

Continuing in the realm of proteins, the EGNN has been applied in a variety of tasks such as antibody structure prediction (Abanades et al., 2022), protein design (Song et al., 2023), protein backbone generation (Trippe et al., 2022; Mahmud et al., 2023; Wu et al., 2021), protein representation learning (Zhang et al., 2022) and protein dynamics simulations (Chen et al., 2023).

Further works include improving molecular Force Fields through transfer learning (Gao et al., 2022; Cui et al., 2023), improving Density Functional Theory predictions through delta-learning (Atz et al., 2022), materials design (Govindarajan et al., 2023), developing more accurate equivariant networks (Brandstetter et al., 2021) and further topics such as robotic kinematics (Limoyo et al., 2023).

Moving to Chapters 6 and 7, we presented Equivariant Generative Models for molecule generation. This family of generative models is starting to have a significant impact on molecular modelling. In particular, some of the works already mentioned in the EGNN paragraphs that generate ligands in molecule-ligand interactions have been adapted from E(n) Equivariant Normalizing Flows introduced in Chapter 6 (Rozenberg and Freedman, 2022) and from the Equivariant Diffusion Models introduced in Chapter 7 (Igashov et al., 2022; Schneuing et al., 2022). More broadly, equivariant generative models are emerging as a new paradigm in the field of chemical sciences (Anstine and Isayev, 2023) and po-

tentially influencing key applications such as drug design (Isert et al., 2023) and material design (Zeni et al., 2023).

Last but not least, the hybrid models discussed in chapters 3 and 4 have influenced subsequent research in model-based deep learning, leading to enhancements in Neural Network Kalman Filters and in signal processing as seen in the works (Revach et al., 2022; Pratik et al., 2020; Shlezinger et al., 2023).

2

BACKGROUND

2.1 NOTATION

This section outlines the main notation used in the thesis. Additional notation will provided as required in different chapters.

Symbol	Description	Example
x, y, z	Scalars are denoted by lowercase italic letters.	$x = 3$
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	Vectors are represented by boldface lowercase letters.	$\mathbf{x} = (x_1, x_2, x_3)$
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	Matrices are denoted by boldface uppercase letters.	$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$
$\mathcal{S}, \mathcal{V}, \mathcal{U}$	Sets are represented by calligraphic uppercase letters.	$\mathcal{S} = \{s_1, s_2, s_3\}$
f, g, ϕ	Functions are denoted by lowercase letters.	$f(x) = x^2$
∇, Δ	Common operators such as the gradient and Laplacian.	$\nabla f, \Delta f$
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.	$\mathcal{N}(\mathbf{0}, \mathbf{I})$
$p(\mathbf{x})$	Probability density function of a random variable $\mathbf{x}$.	$p(\mathbf{x}) \propto e^{-\mathbf{x}^\top \mathbf{x}}$
$\mathbb{R}^N$	N-dimensional Euclidean space.	$\mathbb{R}^3$

Table 1: General notation for scalars, vectors, and matrices used throughout the thesis, including notation for multivariate Gaussian distributions and probability density functions.

The following sections recall some some basic background concepts that are used trough the thesis.

2.2 NEURAL NETWORKS

A neural network in its most basic form is a sequence of linear operations interspersed with non-linear functions also known as activation layers. To be more precise, we can define a neural network ϕ as a function composed of linear operations $f_l(\mathbf{h}) = \mathbf{W}_l\mathbf{h} + \mathbf{b}_l$ and non-linearities such as the sigmoid function $\sigma(x) = 1/(1+e^{-x})$ which operates element-wise, as detailed below:

$$\phi = f_L \circ \sigma \circ f_{L-1} \circ \cdots \circ \sigma \circ f_2 \circ \sigma \circ f_1 \tag{1}$$

where $\mathbf{W}_l \in \mathbb{R}^{m_l \times n_l}$ are the weights of the matrix multiplication and addition term $\mathbf{b}_l \in \mathbb{R}^m_l$ at the linear layer l. Notice that all learnable parameters are contained in the linear layers f_l. The resulting neural network is a function $\phi : \mathbb{R}^{n^0} \rightarrow \mathbb{R}^{m^L}$ that maps from an input space of dimensionality n_0 defined as the number of columns in $\mathbf{W}_0$ to an output space of dimensionality m^L defined as number of rows in $\{\mathbf{W}_L, \mathbf{b}_l\}$. The remaining number of rows in the weight matrices of this neural network are of arbitrary choice (as long as $m_l = n_{l+1}$ for all l). Therefore some of the most common hyperparameters to tune in a neural network are what we usually call the number of hidden neurons per layer m^l for $l < N$ and the number of layers L.

A strength of neural networks is that they are universal approximators for continuous functions (Cybenko, 1989), meaning that a neural network with at least one hidden layer and a sufficient number of neurons can approximate any continuous function. Notice that this characteristic does not guarantee generalization but the capacity to overfit the training data. Therefore, incorporating inductive biases into neural networks is crucial to obtain good generalization performance. Multilayered neural networks as the one defined in this section are also known by the name Multilayer Perceptron (MLP).

2.3 GRAPH NEURAL NETWORKS

Graph Neural Networks are permutation equivariant networks that operate on graph structured data (Bruna et al., 2013; Defferrard et al., 2016; Kipf and Welling, 2016a). Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $v_i \in \mathcal{V}$ and edges $e_{ij} \in \mathcal{E}$ where each edge connect a pair of nodes. Each node v_i has associated a node of features $\mathbf{h}_i \in \mathbb{R}^{nf}$ where "nf" is the number of features.

Then, a Graph Neural Network (GNN) takes as input the edges $\mathcal{E}$ and a set of node features $\mathbf{H}^0 = \{\mathbf{h}_1, \ldots, \mathbf{h}_i, \ldots \mathbf{h}_N\}$ at layer 0 and outputs a transformation $\mathbf{H}^L$ after L layers such as $\mathbf{H}^L = \text{GNN}[\mathbf{H}^0, \mathcal{E}]$. A very important property of GNNs

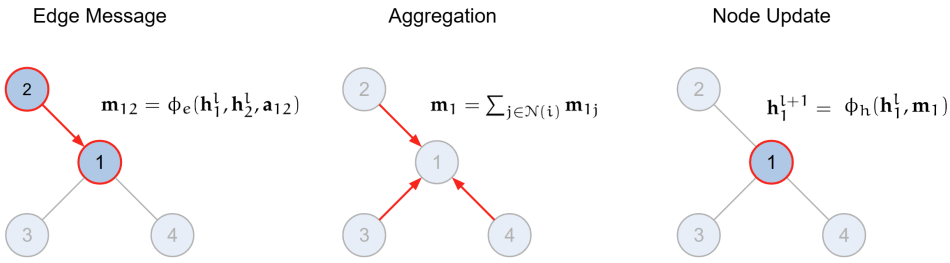

Figure 1: Illustration of the three main Graph Neural Network operations.

not present in Multilayer Perceptrons from previous Section 2.2 is that they are permutation equivariant. In other words, permuting the input results in the same permutation of the output such as $P(\mathbf{H}^L) = \text{GNN}[P(\mathbf{H}^0, \mathcal{E})]$ where P is a permutation operator of the node indexes.

A GNN is usually defined as a sequence of Graph Convolutional Layers (GCL) which are also permutation equivariant. Each GCL takes as input $\mathbf{H}^l$ at a given layer l and outputs an update $\mathbf{H}^{l+1}$ such as $\mathbf{H}^{l+1} = \text{GCL}[\mathbf{H}^l, \mathcal{E}]$. Following the notation from Gilmer et al., 2017 we define a GCL layer as:

$$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{a}_{ij}) \quad \text{Edge message} \tag{2}$$

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij} \quad \text{Aggregation} \tag{3}$$

$$\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i) \quad \text{Node update} \tag{4}$$

where $\mathbf{h}_i^l \in \mathbb{R}^{nf}$ is the nf-dimensional embedding of node v_i at layer l, $\mathbf{a}_{ij}$ are optional edge attributes, $\mathcal{N}(i)$ represents the set of neighbors of node v_i. Finally, ϕ_e and ϕ_h are the edge and node operations respectively which are commonly modelled by Multilayer Perceptrons (MLPs) such as the network introduced in previous Section 2.2. The GCL defined here and illustrated in Figure 1 consists of three main operations :

Edge message: This operation computes a message for each edge (i, j) in the graph, taking as input features of a neighbor node $\mathbf{h}_j^l$ and the node itself $\mathbf{h}_i^l$. Additional edge features can be incorporated if necessary through the vector $\mathbf{a}_{ij}$.

Aggregation: This operation aggregates the incoming message to a node i from all its neighbor nodes $\mathcal{N}(i)$. Typically it is just a sum operation but it could take other forms as long as it satisfies permutation equivariance.

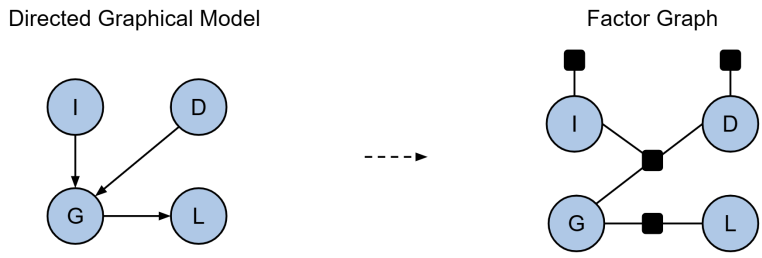

Figure 2: (Left) Example of a very simple directed graphical model. (Right) its representation as a Factor Graph.

Node update: This operation computes an update on the node embeddings given the embeddings $\mathbf{h}_i^l$ and the aggregated message $\mathbf{m}_i$.

2.4 GRAPHICAL MODELS

A Graphical Model is a structured representation of the data generating process, where nodes represent random variables and edges describe the dependence between these random variables. This allows for an efficient representation of complex probability distributions.

Graphical Models can be categorized as directed or undirected depending on whether their edges contain directional information, and also as cyclic or acyclic depending on whether we can find loops in the graph. An example of a Directed Acyclic Graphical Model is shown at the left of Figure 2. This graphical model contains four random variables x_I, x_D, x_G, x_L and its joint distribution can be factorized as $p(x_I, x_D, x_G, x_L) = p(x_I)p(x_D)p(x_G|x_I, x_D)p(x_L|x_G)$. Imagine x_I is a random variable describing the intelligence of a student, x_D the difficulty of an exam, x_G the obtained grade and x_L whether getting or not a recommendation letter. This graphical model shows that for example, given x_G, the variable x_L will be independent from x_I and x_D. Now we can 'reason' over this graphical model to obtain some desired measurements such as the marginal of one of its variables, for example, to obtain the marginal $p(x_L)$ we need to marginalize the other variables $p(x_L) = \sum_I \sum_D \sum_G p(x_I)p(x_D)p(x_G|x_I, x_D)p(x_L|x_G)$ which we can do in two steps, first $p(x_L|x_I, x_D) = \sum_{x_G} p(x_G|x_I, x_D)p(x_L|x_G)$ and then $p(x_L) = \sum_{x_I} \sum_{x_D} p(x_I)p(x_D)p(x_L|x_I, x_D)$.

These two steps can be conceptualized as local operations or messages among subsets of nodes in the graphical model. Even for a small graphical model of 4 nodes, these integrals start to look convoluted. If we want to compute multiple marginals on much larger graphs, coming up with these marginalization rules may become much more complex. To facilitate this, message passing algorithms have been developed to perform inference on graphical model variables, such

as the sum-product algorithm also known as Belief Propagation (Pearl, 1988). Belief Propagation locally marginalizes over random variables. It exploits the structure of factor graphs, allowing more efficient computation of the marginals. We will introduce this algorithm in more detail in Chapter 4. For a graphical model without cycles, Belief Propagation can obtain the true marginals.

2.4.1 *Factor Graphs*

Factor Graphs (Loeliger, 2004) are a convenient way of representing graphical models. A factor graph is a bipartite graph that interconnects a set of factors $f_s(\mathbf{x}_s)$ with a set of variables $\mathbf{x}$. Here, each $\mathbf{x}_s$ is a subset of the variables $\mathbf{x}$, and each factor $f_s(\mathbf{x}_s)$ defines dependencies among its subset of variables $\mathbf{x}_s$. A visual representation of a Factor Graph derived from the previous directed graphical model is depicted at the right of Figure 2, where black squares represent factors and blue circles variables. For this particular example in Figure 2, $\mathbf{x} = \{x_I, x_D, x_G, x_L\}$, and for instance, the factor in the middle of the graph that connects to three variables would be defined as $f_{x_I,x_D,x_G}(x_I, x_D, x_G)$. A global probability distribution $p(\mathbf{x})$ can be defined as the product of all factors in the graphical model $p(\mathbf{x}) = \frac{1}{Z}\prod_{s\in\mathcal{F}} f_s(\mathbf{x}_s)$, where Z is the normalization constant of the probability distribution and $\mathcal{F}$ is the set of all factors. We can re-write the previous graphical model as the factor graph: $p(x_I, x_D, x_G, x_L) = f_{x_I}(x_I)f_{x_D}(x_D)f_{x_G,x_I,x_D}(x_G, x_I, x_D)f_{x_L,x_G}(x_L, x_G)$. Notice that the factor graph notation discards the directional dependencies between variables. This Factor Graph notation will be used Chapter 4.

2.5 EQUIVARIANCE

In simple terms, equivariance is a concept in which the output of a function changes in a consistent way in response to transformations applied to its input. For example, if rotating the input of a function ϕ results in the same rotation at the output, the function is considered equivariant to rotations such as $\mathbf{R}(\phi(\mathbf{X})) = \phi(\mathbf{R}(\mathbf{X}))$, where $\mathbf{R}$ is a rotation operator and $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_M) \in \mathbb{R}^{M\times n}$ an input set of M points in a point cloud embedded in a n-dimensional space.

Formally, equivariance can be defined as follows. Let $T_g : X \to X$ be a set of transformations on X for an abstract group $g \in G$. We say a function $\phi : X \to Y$ is equivariant to g if there exists an equivalent transformation on its output space $S_g : Y \to Y$ such that $\phi(T_g(\mathbf{x})) = S_g(\phi(\mathbf{x}))$. In this thesis, particularly in Chapter 5, 6 and 7, we focus on the Euclidean group E(n) which includes translations, rotations and reflections in addition to also being interested in the permutation group. Specifically, given $\mathbf{X} \in \mathbb{R}^{M\times n}$ and $\phi(\mathbf{X}) = \mathbf{Y} \in \mathbb{R}^{M\times n}$ the transformed set of point clouds, we are interested in the following types of equivariance:

Translation equivariance: Translating the input by $\mathbf{g} \in \mathbb{R}^n$ results in an equivalent translation of the output. Let $\mathbf{X} + \mathbf{g}$ be shorthand for $(\mathbf{x}_1 + \mathbf{g}, \ldots, \mathbf{x}_M + \mathbf{g})$. Then $\mathbf{Y} + \mathbf{g} = \phi(\mathbf{X} + \mathbf{g})$.

Rotation (and reflection) equivariance: For any orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$, let $\mathbf{QX}$ be shorthand for $(\mathbf{Qx}_1, \ldots, \mathbf{Qx}_M)$. Then rotating the input results in an equivalent rotation of the output $\mathbf{QY} = \phi(\mathbf{QX})$.

Permutation equivariance: Permuting the input results in the same permutation of the output $P(\mathbf{Y}) = \phi(P(\mathbf{X}))$ where P is a permutation on the row indexes.

For instance, the forces within a molecule, resulting from its atomistic interactions are rotation equivariant. This implies that if we rotate the Cartesian positions of the molecule in a 3D space, the vector force field rotates accordingly.

2.6 DEEP GENERATIVE MODELS

A Generative Model is a statistical model that is capable of generating new samples from a probability distribution $\mathbf{x} \sim p_\theta(\mathbf{x})$ where θ are the learnable parameters that define the model. With the advance of deep learning, Deep Generative Models have evolved to model much more complex probability distributions. This has lead to very impactful applications in image, text and audio generation (Ramesh et al., 2021; Brown et al., 2020; Oord et al., 2016).

Deep Generative Models aim to model a probability distribution $p(\mathbf{x})$ to which we may not have direct access. However, we often have access to a set of samples drawn from that probability distribution $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_D\}$. These samples can be used to learn a probability distribution as $p_\theta(\mathbf{x}) \approx p(\mathbf{x})$, approximating the true one.

Ideally, to learn p_θ, we would like to optimize a similarity metric between the true and the parametrized distributions with respect to θ. A common choice to measure distribution similarity is the Kullback–Leibler divergence $KL(p, p_\theta) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{p_\theta(\mathbf{x})}$. As shown in the next equations, minimizing the $KL(p, p_\theta)$ between distributions with respect to θ is equivalent to minimizing the cross-entropy $H(p, p_\theta)$:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\ \mathrm{KL}(p, p_\theta) = \underset{\theta}{\operatorname{argmin}} \int p(\mathbf{x}) \log p(\mathbf{x}) - p(\mathbf{x}) \log p_\theta(\mathbf{x}) \tag{5}$$

$$= \underset{\theta}{\operatorname{argmin}} \int - p(\mathbf{x}) \log p_\theta(\mathbf{x}) \tag{6}$$

$$= \underset{\theta}{\operatorname{argmin}}\ \mathrm{H}(p, p_\theta) \tag{7}$$

This is possible because the first term of the KL divergence does not depend on θ, allowing us to discard that term when optimizing with respect to θ. The entropy term $\mathrm{H}(p, p_\theta) = \int -p(\mathbf{x}) \log p_\theta(\mathbf{x})$ can be rewritten as an expectation over samples $\mathbb{E}\mathbf{x} \sim p(\mathbf{x})[-\log p_\theta(\mathbf{x})]$. However, we do not have access to the true distribution $p(\mathbf{x})$, but we can approximate it from a limited set of samples $\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[-\log p_\theta(\mathbf{x})]$ from our dataset. This results in minimizing the negative log-likelihood of the data with respect to the model parameters θ resulting in the following equation to optimize:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[-\log p_\theta(\mathbf{x})] \tag{8}$$

In practice, direct minimization of the negative log-likelihood is not always tractable, especially for some models that introduce latent variables $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}$ such as Variational Autoencoders (Kingma and Welling, 2013) and Diffusion Models (Ho et al., 2020). However, in such cases, we can optimize a variational lower bound on the log-likelihood $\log p_\theta(\mathbf{x})$ (Kingma and Welling, 2013). Different generative models approach the optimization the log-likelihood term or the lower bound in different ways. In this thesis we work with Normalizing Flows and Diffusion Models, the details on how these methods are optimized are introduced in their respective chapters 6 and 7. For both cases we will incorporate E(3) Equivariant constraints into these generative models to adapt them for data existing in a 3-dimensional space.

Part I

HYBRID INFERENCE

3

HYBRID INFERENCE IN HIDDEN MARKOV MODELS

This Chapter is based on the content of:
Victor Garcia Satorras et al. (2019). "Combining generative and discriminative models for hybrid inference." In: *Advances in Neural Information Processing Systems* 32

3.1 INTRODUCTION

Currently, deep learning is a dominant paradigm in the machine learning community. Before that, however, one of the dominant paradigms in machine learning was graphical models (Bishop, 2006; Murphy, 2012; Koller et al., 2009). Graphical models structure the space of random variables by organizing them into a dependency graph. For instance, some variables have parents/children dependencies (directed models) or neighbor dependencies (undirected models). These dependencies are encoded by conditional probabilities (directed models) or potentials (undirected models). While these interactions can have learnable parameters, the structure of the graph imposes a strong inductive bias onto the model. Reasoning in graphical models is performed by a process called probabilistic inference where the posterior distribution, or the most probable state of a set of variables, is computed given observations of other variables. Many approximate algorithms have been proposed to solve this problem efficiently, among which are MCMC sampling (Neal et al., 2011; Salimans et al., 2015), variational inference (Kingma and Welling, 2013) and belief propagation algorithms (Crick and Pfeffer, 2002; Koller et al., 2009).

Graphical models are a kind of generative model where we specify important aspects of the generative process. They excel in the low data regime because we maximally utilize expert knowledge (a.k.a. inductive bias). However, human imagination often falls short of modeling all of the intricate details of the true underlying generative process. In the large data regime there is an alternative strategy which we could call "learning to infer". Here, we create lots of data pairs $\{\mathbf{x}_n, \mathbf{y}_n\}$ with $\{\mathbf{y}_n\}$ the observed variables and $\{\mathbf{x}_n\}$ the latent unobserved random variables. These can be generated from the generative model or are avail-

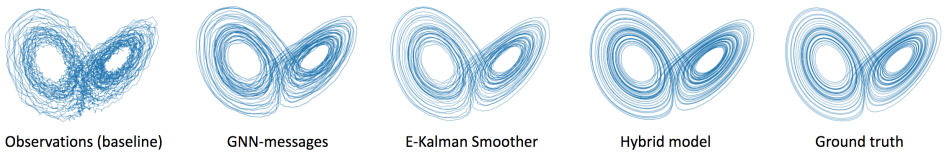

Figure 3: Examples of inferred 5K length trajectories for the Lorenz attractor with $\Delta t = 0.01$ trained on 50K length trajectory. The mean squared errors from left to right are (Observations: 0.2462, GNN: 0.0613, E-Kalman Smoother: 0.0372, Hybrid: 0.0169).

able directly in the dataset. Our task is now to learn a flexible mapping $q(\mathbf{x}|\mathbf{y})$ to infer the latent variables directly from the observations. This idea is known as "inverse modeling" in some communities. It is also known as "amortized" inference (Rezende and Mohamed, 2015a) or recognition networks in the world of variational autoencoders (Kingma and Welling, 2013) and Helmholtz machines (Dayan et al., 1995).

In this chapter we consider inference as an iterative message passing scheme over the edges of the graphical model. We know that (approximate) inference in graphical models can be formulated as message passing, known as belief propagation, so this is a reasonable way to structure our computations. When we unroll these messages for N steps we have effectively created a recurrent neural network as our computation graph. We will enrich the traditional messages with a learnable component that has the function to correct the original messages when there is enough data available. In this way we create a hybrid message passing scheme with prior components from the graphical model and learned messages from data. The learned messages may be interpreted as a kind of graph convolutional neural network (Bruna et al., 2013; Henaff et al., 2015; Kipf and Welling, 2016a).

Our hybrid model neatly trades off the benefit of using inductive bias in the small data regime and the benefit of a much more flexible and learnable inference network when sufficient data is available. In this chapter we restrict ourselves to a sequential model known as a hidden Markov process.

3.2 THE HIDDEN MARKOV PROCESS

In this section we briefly explain the Hidden Markov Process and how we intend to extend it. In a Hidden Markov Model (HMM), a set of unobserved variables $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_K\}$ define the state of a process at every time step $0 < k < K$. The set of observable variables from which we want to infer the process states are denoted by $\mathbf{Y} = \{\mathbf{y}_1, \dots \mathbf{y}_K\}$. HMMs are used in diverse applications as localization, tracking, weather forecasting and computational finance among others. (in fact, the Kalman filter was used to land the Eagle on the moon.)

We can express $p(\mathbf{X}|\mathbf{Y})$ as the probability distribution of the hidden states given the observations. Our goal is to find which states $\mathbf{x}$ maximize this probability distribution. More formally:

$$\hat{\mathbf{X}} = arg\,max_{\mathbf{X}} p(\mathbf{X}|\mathbf{Y}) \tag{9}$$

Under the Markov assumption i) the transition model is described by the transition probability $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, and ii) the measurement model is described by $p(\mathbf{y}_t|\mathbf{x}_t)$. Both distributions are stationary for all k. The resulting graphical model can be expressed with the following equation:

$$p(\mathbf{X}, \mathbf{Y}) = p(\mathbf{x}_0) \prod_{k=1}^{K} p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{y}_k|\mathbf{x}_k) \tag{10}$$

One of the best known approaches for inference problems in this graphical model are the Kalman Filter (Kalman, 1960) and Smoother (Rauch et al., 1965). The Kalman Filter assumes both the transition and measurement distributions are linear and Gaussian. The prior knowledge we have about the process is encoded in linear transition and measurement processes, and the uncertainty of the predictions with respect to the real system is modeled by Gaussian noise:

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{q}_k \tag{11}$$

$$\mathbf{y}_k = \mathbf{H}\mathbf{x}_k + \mathbf{r}_k \tag{12}$$

Here $\mathbf{q}_k, \mathbf{r}_k$ come from Gaussian distributions $\mathbf{q}_k \sim \mathcal{N}(0, \mathbf{Q})$, $\mathbf{r}_k \sim \mathcal{N}(0, \mathbf{R})$. $\mathbf{F}$, $\mathbf{H}$ are the linear transition and measurement functions respectively. If the process from which we are inferring $\mathbf{x}$ is actually Gaussian and linear, a Kalman Filter + Smoother with the right parameters is able to infer the optimal state estimates.

The real world is usually non-linear and complex, assuming that a process is linear may be a strong limitation. Some alternatives like the Extended Kalman Filter (Ljung, 1979) and the Unscented Kalman Filter (Wan and Van Der Merwe, 2000) are used for non-linear estimation, but even when functions are non-linear, they are still constrained to our knowledge about the dynamics of the process which may differ from real world behavior.

To model the complexities of the real world we intend to learn them from data through flexible models such as neural networks. In this work we present an hybrid inference algorithm that combines the knowledge from a generative model (e.g. physics equations) with a function that is automatically learned from data using a neural network. In our experiments we show that this hybrid method outperforms the graphical inference methods and also the neural network methods for low and high data regimes respectively. In other words, our method benefits from the inductive bias in the limit of small data and also the high capacity of

a neural networks in the limit of large data. The model is shown to gracefully interpolate between these regimes.

3.3 RELATED WORK

The proposed method has interesting relations with meta learning (Andrychowicz et al., 2016) since it learns more flexible messages on top of an existing algorithm. It is also related to structured prediction energy networks (Belanger et al., 2017) which are discriminative models that exploit the structure of the output. Structured inference in relational outputs has been effective in a variety of tasks like pose estimation (Wei et al., 2016), activity recognition (Deng et al., 2016) or image classification (Nauata et al., 2018). One of the closest works is Recurrent Inference Machines (RIM) (Putzky and Welling, 2017) where a generative model is also embedded into a Recurrent Neural Network (RNN). However in that work graphical models played no role. In the same line of learned recurrent inference, our optimization procedure shares similarities with Iterative Amortized Inference (Marino et al., 2018), although in our work we are refining the gradient using a hybrid setting while they are learning it.

Another related line of research is the convergence of graphical models with neural networks, (Mirowski and LeCun, 2009) replaced the joint probabilities with trainable factors for time series data. Learning the messages in conditional random fields has been effective in segmentation tasks (Chen et al., 2014; Zheng et al., 2015). Relatedly, Johnson et al. (2016) runs message passing algorithms on top of a latent representation learned by a deep neural network. More recently (Yoon et al., 2018) showed the efficacy of using Graph Neural Networks (GNNs) for inference on a variety of graphical models, and compared the performance with classical inference algorithms. This last work is in a similar vein as ours, but in our case, learned messages are used to correct the messages from graphical inference. In the experiments we will show that this hybrid approach really improves over running GNNs in isolation.

The Kalman Filter is a widely used algorithm for inference in Hidden Markov Processes. Some works have explored the direction of coupling them with machine learning techniques. A method to discriminatively learn the noise parameters of a Kalman Filter was introduced by Abbeel et al. (2005). In order to input more complex variables, Haarnoja et al. (2016) back-propagates through the Kalman Filter such that an encoder can be trained at its input. Similarly, Coskun et al. (2017) replaces the dynamics defined in the Kalman Filter with a neural network. In our hybrid model, instead of replacing the already considered dynamics, we simultaneously train a learnable function for the purpose of inference.

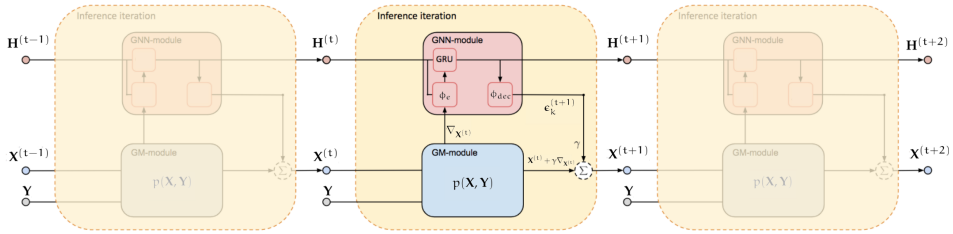

Figure 4: Graphical illustration of our Hybrid algorithm. The GM-module (blue box) sends messages to the GNN-module (red box) which refines the estimation of $\mathbf{X}$.

3.4 MODEL

We cast our inference model as a message passing scheme where the nodes of a probabilistic graphical model can send messages to each other to infer estimates of the states $\mathbf{x}$. Our aim is to develop a hybrid scheme where messages derived from the generative graphical model are combined with GNN messages:

Graphical Model Messages (GM-messages): These messages are derived from the generative graphical model (e.g. equations of motion from a physics model).

Graph Neural Network Messages (GNN-messages): These messages are learned by a GNN which is trained to reduce the inference error on labelled data in combination with the GM-messages.

In the following two subsections we introduce the two types of messages and the final hybrid inference scheme.

3.4.1 *Graphical Model Messages*

In order to define the GM-messages, we interpret inference as an iterative optimization process to estimate the maximum likelihood values of the states $\mathbf{X}$. In its more generic form, the recursive update for each consecutive estimate of $\mathbf{X}$ is given by:

$$\mathbf{X}^{(t+1)} = \mathbf{X}^{(t)} + \gamma \nabla_{\mathbf{X}^{(t)}} \log(p(\mathbf{X}^{(t)}, \mathbf{Y})) \tag{13}$$

Factorizing Equation 13 to the hidden Markov Process from Equation 10, we get three input messages for each inferred node $\mathbf{x}_k$:

$$\mathbf{x}_k^{(t+1)} = \mathbf{x}_k^{(t)} + \gamma \boldsymbol{\mu}_k^{(t)} \tag{14}$$

$$\boldsymbol{\mu}_k^{(t)} = \boldsymbol{\mu}_{\mathbf{x}_{k-1}\to\mathbf{x}_k}^{(t)} + \boldsymbol{\mu}_{\mathbf{x}_{k+1}\to\mathbf{x}_k}^{(t)} + \boldsymbol{\mu}_{\mathbf{y}_k\to\mathbf{x}_k}^{(t)}$$

$$\boldsymbol{\mu}_{\mathbf{x}_{k-1}\to\mathbf{x}_k}^{(t)} = \frac{\partial}{\partial \mathbf{x}_k^{(t)}} \log(p(\mathbf{x}_k^{(t)}|\mathbf{x}_{k-1}^{(t)})) \tag{15}$$

$$\boldsymbol{\mu}_{\mathbf{x}_{k+1}\to\mathbf{x}_k}^{(t)} = \frac{\partial}{\partial \mathbf{x}_k^{(t)}} \log(p(\mathbf{x}_{k+1}^{(t)}|\mathbf{x}_k^{(t)})) \tag{16}$$

$$\boldsymbol{\mu}_{\mathbf{y}_k\to\mathbf{x}_k}^{(t)} = \frac{\partial}{\partial \mathbf{x}_k^{(t)}} \log(p(\mathbf{y}_k|\mathbf{x}_k^{(t)})) \tag{17}$$

These messages can be obtained by computing the three derivatives from equations 15, 16, 17. It is often assumed that the transition and measurement distributions $p(\mathbf{x}_k|\mathbf{x}_{k-1})$, $p(\mathbf{y}_k|\mathbf{x}_k)$ are linear and Gaussian (e.g. Kalman Filter model). Next, we provide the expressions of the GM-messages when assuming the linear and Gaussian functions from equations 11, 12:

$$\boldsymbol{\mu}_{\mathbf{x}_{k-1}\to\mathbf{x}_k} = -\mathbf{Q}^{-1}(\mathbf{x}_k - \mathbf{F}\mathbf{x}_{k-1}) \tag{18}$$

$$\boldsymbol{\mu}_{\mathbf{x}_{k+1}\to\mathbf{x}_k} = \mathbf{F}^T\mathbf{Q}^{-1}(\mathbf{x}_{k+1} - \mathbf{F}\mathbf{x}_k) \tag{19}$$

$$\boldsymbol{\mu}_{\mathbf{y}_k\to\mathbf{x}_k} = \mathbf{G}^T\mathbf{R}^{-1}(\mathbf{y}_k - \mathbf{G}\mathbf{x}_k) \tag{20}$$

3.4.2 *Adding GNN-messages*

We call $\mathbf{V}$ the collection of nodes of the graphical model $\mathbf{V} = \mathbf{X} \cup \mathbf{Y}$. We also define an analogous graph where the GNN operates by propagating the GNN messages. We build the following mappings from the nodes of the graphical model to the nodes of the GNN: $\mathbf{H_x} = \{\psi(\mathbf{x}) : \mathbf{x} \in \mathbf{X}\}$, $\mathbf{H_y} = \{\psi(\mathbf{y}) : \mathbf{y} \in \mathbf{Y}\}$. Analogously, the union of both collections would be $\mathbf{H_v} = \mathbf{H_x} \cup \mathbf{H_y}$. Therefore, each node of the graphical model has a corresponding node $\mathbf{h}$ in the GNN. The edges for both graphs are also equivalent. Values of $\mathbf{H}_\mathbf{x}^{(0)}$ that correspond to unobserved variables $\mathbf{X}$ are randomly initialized. Instead, values $\mathbf{H}_\mathbf{y}^{(0)}$ are obtained by forwarding each $\mathbf{y}_k$ through a linear layer. The elements within these sets are denoted $\mathbf{h_v} \in \mathbf{H_v}$ and $\mathbf{h_x} \in \mathbf{H_x}$.

Next we present the equations of the learned messages, which consist of a GNN message passing operation. Similarly to (Li et al., 2015; Kipf et al., 2018), a GRU

(Chung et al., 2014) is added to the message passing operation to make it recursive:

$$\mathbf{m}_{k,n}^{(t)} = \phi_e(\mathbf{h}_{x_k}^{(t)}, \mathbf{h}_{v_n}^{(t)}, \boldsymbol{\mu}_{v_n \to x_k}^{(t)}) \quad \text{Edge message} \tag{21}$$

$$\mathbf{m}_{k}^{(t)} = \sum_{v_n \in \mathcal{N}(x_k)} \mathbf{m}_{k,n}^{(t)} \quad \text{Aggregation over edges} \tag{22}$$

$$\mathbf{h}_{x_k}^{(i+1)} = \text{GRU}(\mathbf{m}_{k}^{(t)}, \mathbf{h}_{x_k}^{(t)}) \quad \text{Node update} \tag{23}$$

$$\boldsymbol{\epsilon}_{k}^{(t+1)} = \phi_{dec}(\mathbf{h}_{x_k}^{(t+1)}) \quad \text{Correction factor} \tag{24}$$

Each GNN message $\mathbf{m}_{k,n}^{(t)}$ is computed by the function $\phi_e(\cdot)$, which receives as input two hidden states from the previous recurrent iteration, and their corresponding GM-message $\boldsymbol{\mu}_{v_n \to x_k}$, this function is approximated by a different neural network for the two types of edges (e.g. transition or measurement in the Hidden Markov Model from Equation 10). Equation 22 aggregates all incoming messages in the neighborhood set $\mathcal{N}(\mathbf{x}_k)$, this sum of messages is represented as $\mathbf{m}_k^{(t)}$. The sum of messages $\mathbf{m}_k^{(t)}$ is then provided as input to the GRU neural network that updates each hidden state $\mathbf{h}_{x_k}^{(t)}$ for each node. The GRU is composed by a single GRU cell preceded by a linear layer at its input. Finally a correction signal $\boldsymbol{\epsilon}_k^{(t+1)}$ is decoded from each hidden state $\mathbf{h}_{x_k}^{(t+1)}$ and it is added to the recursive operation 14, resulting in the final equation:

$$\mathbf{x}_k^{(t+1)} = \mathbf{x}_k^{(t)} + \gamma(\boldsymbol{\mu}_k^{(t)} + \boldsymbol{\epsilon}_k^{(t+1)}) \tag{25}$$

In summary, Equation 25 defines our hybrid model in a simple recursive form where $\mathbf{x}_k$ is updated through two contributions: one that relies on the probabilistic graphical model messages $\boldsymbol{\mu}_k^{(t)}$, and $\boldsymbol{\epsilon}_k^{(t)}$, that is automatically learned. We note that it is important that the GNN messages model the *"residual error"* of the GM inference process, which is often simpler than modeling the full signal. A visual representation of the algorithm is shown in Figure 2.

In the experimental section of this work we apply our model to the Hidden Markov Process, however, the above mentioned GNN-messages are not constrained to this particular graphical structure. The GM-messages can also be obtained for other arbitrary graph structures by applying the recursive inference equation 13 to their respective graphical models.

3.4.3 *Training Procedure*

The loss function is computed at every iteration, with a weighted sum that emphasizes later iterations, $w_i = \frac{i}{N}$, more formally:

$$\text{Loss}(\Theta) = \sum_{i=1}^{N} w_i \mathcal{L}(\mathbf{gt}, \Gamma(\mathbf{x}^{(t)})) \tag{26}$$

In this equation, the function $\Gamma(\cdot)$ is designed to extract the elements of the hidden state $\mathbf{x}$ that overlap with the ground truth $\mathbf{gt}$. To give a more concrete example, imagine $\mathbf{x}$ contains positions, velocity, and acceleration. Out of these, in many tracking systems, we may only have the ground truth for positions, while the rest is inferred by model. The function $\Gamma(\cdot)$ would extract the positions from $\mathbf{x}$ to be compared against the ground truth. In our experiments we choose the mean square error as the loss $\mathcal{L}(\cdot)$.

The training procedure consists of three main steps. First, we initialize $\mathbf{x}_k^{(0)}$ at the value that maximizes $p(\mathbf{y}_k|\mathbf{x}_k)$. For example, in a trajectory estimation problem where $p(\mathbf{y}_k|\mathbf{x}_k)$ is approximated as gaussian we initialize the position values of $\mathbf{x}_k^{(0)}$ as the observed positions $\mathbf{y}_k$. Second, we tune/sweep over the hyperparameters of the graphical model as it would be done with a Kalman Filter, which are usually the variance of Gaussian distributions $\mathbf{Q}$ from Equation 12. Finally, we train the proposed hybrid model using the above mentioned loss.

3.5 EXPERIMENTS

In this section we compare our Hybrid model with the Kalman Smoother and a recurrent GNN. We show that our Hybrid model can leverage the benefits of both methods for different data regimes. Next we define the models used in the experiments

Kalman Smoother: The Kalman Smoother is the widely known Kalman Filter algorithm (Kalman, 1960) + the RTS smoothing step (Rauch et al., 1965). In experiments where the transition function is non-linear we use the Extended Kalman Filter + smoothing step which we will call *"E-Kalman Smoother"*.

GM-messages: As a special case of our Hybrid model we propose to remove the learned signal $\boldsymbol{\epsilon}_k^{(t)}$ and base our predictions only on the graphical model messages from Eq. 14.

GNN-messages: The GNN model is another special case of our model when all the GM-messages are removed and only GNN messages are propagated. Instead of decoding a refinement for the current $\mathbf{x}_k^{(t)}$ estimate, we directly esti-

mate: $\mathbf{x}_k^{(t)} = \mathbf{G}^\top \mathbf{y}_k + \phi_{dec}(\mathbf{h}_{\mathbf{x}_k}^{(t)})$. The resulting algorithm is equivalent to a Gated Graph Neural Network (Li et al., 2015).

Hybrid model: This is the name used for our proposed model explained in Section 3.4.2.

For the training settings, we set $\gamma = 0.005$ from Equation 25 and use the Adam optimizer with a learning rate 10^{-3}. The number of inference iterations used in the Hybrid model, GNN-messages and GM-messages is N=50. ϕ_e and ϕ_{dec} are a 2-layers MLPs with Leaky Relu and Relu activations (Nair and Hinton, 2010) respectively. The number of features in the hidden layers of the GRU, ϕ_e and ϕ_{dec} is nf=48. In trajectory estimation experiments, $\mathbf{y}_k$ values may take any value from the real numbers $\mathbb{R}$. Shifting a trajectory to a non-previously seen position may hurt the generalization performance of the neural network. To make the problem translation invariant we modify $\mathbf{y}_k$ before mapping it to $\mathbf{h}_{\mathbf{y}_k}$, we use the difference between the observed current position with the previous one and with the next one.

3.5.1 *Linear Dynamics*

The aim of this experiment is to infer the position of every node in trajectories generated by linear and gaussian equations. The advantage of using a synthetic environment is that we know in advance the original equations the motion pattern was generated from, and by providing the right linear and gaussian equations to a Kalman Smoother we can obtain the optimal inferred estimate as a lower bound of the test loss.

Among other tasks, Kalman Filters are used to refine the noisy measurement of GPS systems. A physics model of the dynamics can be provided to the graphical model that, combined with the noisy measurements, gives a more accurate estimation of the position. The real world is usually more complex than the equations we may provide to our graphical model, leading to a gap between the assumed dynamics and the real world dynamics. Our Hybrid model is able to fill this gap without the need to learn everything from scratch.

To show that, we generate synthetic trajectories $\mathcal{T} = \{\mathbf{X}, \mathbf{Y}\}$. Each state $\mathbf{x}_k \in \mathbb{R}^6$ is a 6-dimensional vector that encodes position, velocity and acceleration $(\mathbf{p}, \mathbf{v}, \mathbf{a})$ for two dimensions. Each $\mathbf{y}_k \in \mathbb{R}^2$ is a noisy measurement of the position also for two dimensions. The transition dynamic is a non-uniform accelerated motion that also considers drag (air resistance):

$$\frac{\partial \mathbf{p}}{\partial t} = \mathbf{v}, \qquad \frac{\partial \mathbf{v}}{\partial t} = \mathbf{a} - c\mathbf{v}, \qquad \frac{\partial \mathbf{a}}{\partial t} = -\tau\mathbf{v} \tag{27}$$

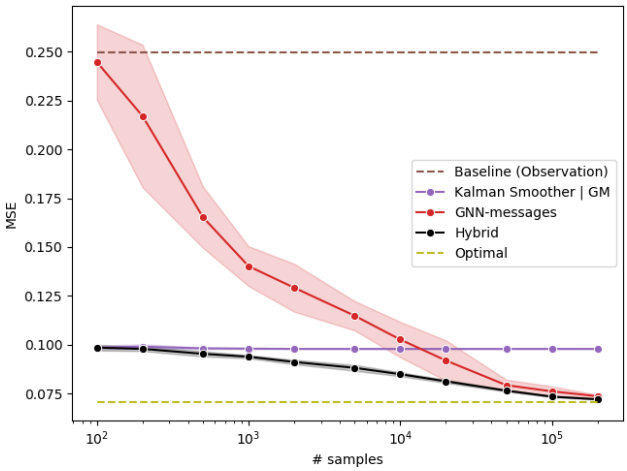

Figure 5: MSE comparison with respect to the number of training samples for the linear dynamics dataset.

Where $-c\mathbf{v}$ represents the air resistance (Falkovich, 2011), with c being a constant that depends on the properties of the fluid and the object dimensions. Finally, the variable $-\tau\mathbf{v}$ is used to non-uniformly accelerate the object.

To generate the dataset, we sample from the Markov process of equation 10 where the transition probability distribution $p(\mathbf{x}_{k+1}|\mathbf{x}_k)$ and the measurement probability distribution $p(\mathbf{y}_k|\mathbf{x}_k)$ follow equations (11, 12). Values $\mathbf{F}, \mathbf{Q}, \mathbf{G}, \mathbf{R}$ for these distributions are described in the Appendix A.1, in particular, $\mathbf{F}$ is analytically obtained from the above mentioned differential equations 27. We sample two different motion trajectories from 50 to 100K time steps each, one for validation and the other for training. An additional 10K time steps trajectory is sampled for testing. The sampling time step is $\Delta t = 1$.

Alternatively, the graphical model of the algorithm is limited to a uniform motion pattern $\mathbf{p} = \mathbf{p}_0 + \mathbf{v}t$. Its equivalent differential equations form would be $\frac{\partial \mathbf{p}}{\partial t} = \mathbf{v}$. Notice that the air friction is not considered anymore and velocity and acceleration are assumed to be uniform. Again the parameters for the matrices $\mathbf{F}, \mathbf{Q}, \mathbf{G}, \mathbf{R}$ when considering a uniform motion pattern are analytically obtained and described in Appendix A.1.

RESULTS. The Mean Square Error with respect to the number of training samples is shown for different algorithms in Figure 5. The plot shows the average and the standard deviation over 7 runs, the sampled test trajectory remains the same over all runs, this is not the case for the training and validation sampled trajectories. Note that the MSE of the Kalman Smoother and GM-messages overlap in the plot since both errors were exactly the same.

Our model outperforms both the GNN or Kalman Smoother in isolation in all data regimes, and it has a significant edge over the Kalman Smoother when the number of samples is larger than 1K. This shows that our model is able to ensemble the advantages of prior knowledge and deep learning in a single framework. These results show that our Hybrid model benefits from the inductive bias of the graphical model equations when data is scarce, and simultaneously it benefits from the flexibility of the GNN when data is abound.

A clear trade-off can be observed between the Kalman smoother and the GNN. The Kalman Smoother clearly performs better for low data regimes, while the GNN outperforms it for larger amounts of data (>10K). The Hybrid model is able to benefit from the strengths of both.

3.5.2 *Lorenz Attractor*

The Lorenz equations describe a non-linear chaotic system used for atmospheric convection. Learning the dynamics of this chaotic system in a supervised way is expected to be more challenging than for linear dynamics, making it an interesting evaluation of our Hybrid model. A Lorenz system is modelled by three differential equations that define the convection rate, the horizontal temperature variation and the vertical temperature variation of a fluid:

$$\frac{\partial z_1}{\partial t} = 10(z_2 - z_1), \qquad \frac{\partial z_2}{\partial t} = z_1(28 - z_3) - z_2, \qquad \frac{\partial z_3}{\partial t} = z_1 z_2 - \frac{8}{3} z_3 \tag{28}$$

To generate a trajectory we run the Lorenz equations with a $\partial t = 10^{-5}$ from which we sample with a time step of $\Delta t = 0.05$ resulting in a single trajectory of 104K time steps. Each point is then perturbed with gaussian noise of standard deviation $\lambda = 0.5$. From this trajectory, 4K time steps are separated for testing, the remaining trajectory of 100K time steps is equally split between training and validation partitions.

Assuming $\mathbf{x} \in \mathbb{R}^3$ is a 3-dimensional vector $\mathbf{x} = [z_1, z_2, z_3]^\top$, we can write down the dynamics matrix of the system as $\mathbf{A}_{|\mathbf{x}}$ from the Lorenz differential eq. 28, and obtain the transition function $\mathbf{F}_{|\mathbf{x}_k}$ (Labbe, 2014) using the Taylor Expansion.

$$\dot{\mathbf{x}} = \mathbf{A}_{|\mathbf{x}}\mathbf{x} = \begin{bmatrix} -10 & 10 & 0 \\ 28 - z_3 & -1 & 0 \\ z_2 & 0 & -\frac{8}{3} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}, \quad \mathbf{F}_{|\mathbf{x}_k} = \mathbf{I} + \sum_{j=1}^{J} \frac{(\mathbf{A}_{|\mathbf{x}_k} \Delta t)^j}{j!} \tag{29}$$

where $\mathbf{I}$ is the identity matrix and J is the number of terms from the Taylor expansion. We run simulations for J=1, J=2 and J=5. For larger J the improvement was minimal. For the measurement model $\mathbf{G} = \mathbf{I}$ we use the identity matrix. For

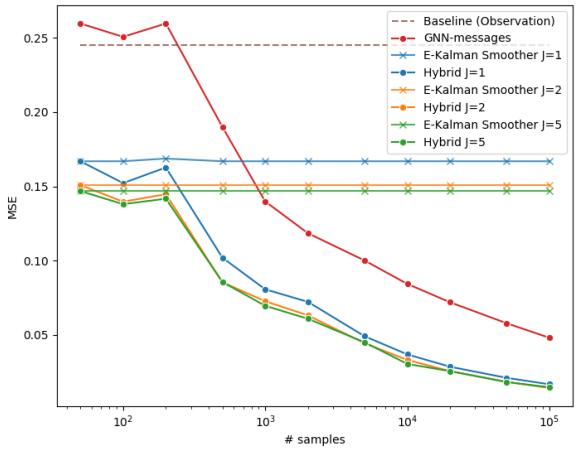

Figure 6: MSE with respect to the the number of training samples on the Lorenz Attractor.

the noise distributions $\mathbf{Q} = \sigma^2 \Delta t \mathbf{I}$ and $\mathbf{R} = 0.5^2 \mathbf{I}$ we use diagonal matrices. The only hyper-parameter to tune from the graphical model is σ.

Since the dynamics are non-linear, the matrix $\mathbf{F}_{|\mathbf{x}_k}$ depends on the values $\mathbf{x}_k$. The presence of these variables inside the matrix introduces a simple non-linearity that makes the function much harder to learn.

RESULTS. The results in Figure 6 show that the GNN struggles to achieve low accuracies for this chaotic system, i.e. it does not converge together with the Hybrid model even when the training dataset contains up to 10^5 samples and the hybrid loss is already $0.01 \sim 0.02$. We attribute this difficulty to the fact the matrix $\mathbf{F}_{|\mathbf{x}_k}$ is different at every state $\mathbf{x}_k$, becoming harder to approximate.

This behavior is different from the previous experiment (linear dynamics) where both the Hybrid model and the GNN converged to the optimal solution for high data regimes. In this experiment, even when the GNN and the E-Kalman Smoother perform poorly, the Hybrid model gets closer to the optimal solution, outperforming both of them in isolation. This shows that the Hybrid model benefits from the labeled data even in situations where its fully-supervised variant or the E-Kalman Smoother are unable to properly model the process. One reason for this could be that the residual dynamics (i.e. the error of the E-Kalman Smoother) are much more linear than the original dynamics and hence easier to model by the GNN.

As can be seen in Figure 6, depending on the amount of prior knowledge used in our Hybrid model we will need more or less samples to achieve a particular accuracy. Following, we show in Table 2 the approximate number of training

samples required to achieve accuracies 0.1 and 0.05 depending on the amount of 'knowledge' we provide (i.e. the number of J terms in the Taylor expansion). The hybrid method requires ~ 10 times less samples than the fully-learned method for MSE=0.1 and ~ 20 times less samples for MSE=0.05.

Table 2: Number of samples required to achieve a given MSE depending on the amount of prior knowledge (i.e. J). These numbers have been extracted from Figure 6.

	GNN ($J = 0$)	Hybrid ($J = 1$)	Hybrid ($J = 2$ & $J = 5$)
MSE = 0.1	~ 5.000	~ 500	~ 400
MSE = 0.05	~ 90.000	~ 5.000	~ 4.000

Finally, qualitative results of estimated trajectories by the different algorithms on the Lorenz attractor are depicted in Figure 3. The plots correspond to a 5K length test trajectory (with $\Delta t = 0.01$). All trainable methods in this plot have been trained on 5K length trajectories.

3.5.3 *Real World Dynamics: Michigan NCLT Dataset*

To demonstrate the generalizability of our Hybrid model to real world data, we use the Michigan NCLT (Carlevaris-Bianco et al., 2016) dataset which is collected by a segway robot moving around the University of Michigan, North Campus. It comprises different trajectories where the GPS measurements and the ground truth location of the robot are provided. Given these noisy GPS observations, our goal is to infer a more accurate position of the segway at a given time.

In our experiments we arbitrarily use the session with date 2012-01-22 which consists of a single trajectory of 6.1 Km on a cloudy day. Sampling at 1Hz results in 4.629 time steps and after removing the parts with an unstable GPS signal, 4.344 time steps remain. Finally, we split the trajectory into three sections: 1.502 time steps for training, 1.469 for validation and 1.373 for testing. The GPS measurements are assumed to be the noisy measurements denoted by $\mathbf{y}$.

Algorithm	MSE
Observations (Baseline)	3.4974
Kalman Smoother	3.0099
GM-Messages	3.0048
GNN-Messages	1.7929
Hybrid model	**1.4109**

Table 3: MSE for different methods on the Michigan NCLT datset.

For the transition and measurement graphical model distributions we assume the same uniform motion model used in Section 3.5.1, specifically the dynamics of a uniform motion pattern. The only parameters to learn from the graphical

model will be the variance of the measurement and transition distributions. The detailed equations are presented in Appendix A.1.1.

RESULTS. Our results show that our Hybrid model (1.4109 MSE) outperforms the GNN (1.7929 MSE), the Kalman Smoother (3.0099 MSE) and the GM-messages (3.0048 MSE). One of the advantages of the GNN and the Hybrid methods on real-world datasets is that both can model the correlations through time from the noise distributions while the GM-messages and the Kalman Smoother assume the noise to be uncorrelated through time as it is defined in the graphical model. In summary, this experiment shows that our Hybrid model can generalize with good performance to a real-world dataset.

3.6 DISCUSSION

In this work, we explored the combination of recent advances in neural networks (e.g., Graph Neural Networks) with more traditional methods of graphical inference in hidden Markov models for time series. The result is a hybrid algorithm that benefits from the inductive bias of graphical models and from the high flexibility of neural networks. We demonstrated these benefits in three different tasks for trajectory estimation: a linear dynamics dataset, a non-linear chaotic system (Lorenz attractor) and a real-world positioning system. In all three experiments, the hybrid method learns to efficiently combine graphical inference with learned inference, outperforming both when run in isolation.

Possible future directions include applying our idea to other generative models (e.g. different graph structures or discrete data). The equations that describe our hybrid model are defined on edges and nodes, therefore, by modifying the input graph, i.e. by modifying the edges and nodes of the input graph, we can run our algorithm on arbitrary graph structures. Furthermore, hybrid models like the one presented in this chapter can help improve the interpretability of model predictions due to their graphical model backbone.

In the next chapter, we will expand upon the ideas presented here to build a hybrid model for graphical models with discrete variables. A conventional approach to perform inference over graphical models with discrete random variables is Belief Propagation; therefore, we will propose a hybrid model to enhance Belief Propagation with Neural Networks.

4

NEURAL ENHANCED BELIEF PROPAGATION

This Chapter is based on the content of:
Victor Garcia Satorras and Max Welling (2021). "Neural enhanced belief propagation on factor graphs." In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 685–693

4.1 INTRODUCTION

In the previous Chapter we presented a Hybrid model for Hidden Markov Models that leveraged ideas from Kalman Filters and Graph Neural Networks. This chapter takes those concepts further to create a more generic hybrid model for graphical models with discrete random variables. Typically, Belief Propagation, that we will introduce later in more detail, is used to reason over discrete graphical models. In this chapter we enhance the capabilities of Belief Propagation with graph neural networks, leading to the model name: Neural Enhanced Belief Propagation.

Graphical models (Bishop, 2006; Murphy, 2012) are a structured representation of locally dependent random variables, that combine concepts from probability and graph theory. A standard way to reason over these random variables is to perform inference on the graphical model using message passing algorithms such as Belief Propagation (BP) (Pearl, 2014; Murphy et al., 2013). Provided that the true generative process of the data is given by a non-loopy graphical model, BP is guaranteed to compute the optimal (posterior) marginal probability distributions. However, in real world scenarios, we may only have access to a poor approximation of the true generative process, leading to sub-optimal estimates. In addition, an important limitation of belief propagation is that on graphs with loops BP computes an approximation to the desired posterior marginals or may fail to converge at all.

In this chapter we present a hybrid inference model to tackle these limitations. We cast our model as a message passing method on a factor graph that combines messages from BP and from a Graph Neural Network (GNN). The GNN

messages are learned from data and complement the BP messages. The GNN receives as input the messages from BP at every inference iteration and delivers as output a refined version of them back to BP. As a result, given a labeled dataset, we obtain a more accurate algorithm that outperforms either Belief Propagation or Graph Neural Networks when run in isolation in cases where Belief Propagation is not guaranteed to obtain the optimal marginals.

Belief Propagation has demonstrated empirical success in a variety of applications: Error correction decoding algorithms (McEliece et al., 1998), combinatorial optimization in particular graph coloring and satisfiability (Braunstein and Zecchina, 2004), inference in Markov logic networks (Richardson and Domingos, 2006), the Kalman Filter is a special case of the BP algorithm (Yedidia et al., 2003; Welch, Bishop, et al., 1995) etc. One of its most successful applications is Low Density Parity Check codes (LDPC) (Gallager, 1962) an error correction decoding algorithm that runs BP on a loopy bipartite graph. LDPC is currently part of the Wi-Fi 802.11 standard, it is an optional part of 802.11n and 802.11ac, and it has been adopted for 5G, the fifth generation wireless technology that began wide deployment in 2019. Despite being a loopy algorithm, its bipartite graph is typically very sparse which reduces the number of loops and increases the cycle size. As a result, in practice LDPC has shown excellent results in error correction decoding tasks and performs close to the Shannon limit in Gaussian channels.

However, a Gaussian channel is an approximation of the more complex noise distributions we encounter in the real world. Many of these distributions have no analytical form, but we can approximate them from data. In this chapter we show the robustness of our algorithm over LDPC codes when we assume such a non-analytical form. Our hybrid method is able to closely match the performance of LDPC in Gaussian channels while outperforming it for deviations from this assumption (i.e. a bursty noise channel (Gilbert, 1960; Kim et al., 2018)).

The three main contributions in this chapter are: i) We extend the standard graph neural network equations to factor graphs (FG-GNN). ii) We present a new hybrid inference algorithm, Neural Enhanced Belief Propagation (NEBP) that refines BP messages using the FG-GNN. iii) We apply our method to an error correction decoding problem for a non-Gaussian (bursty) noise channel and show clear improvement on the Bit Error Rate over existing LDPC codes.

4.2 BACKGROUND

4.2.1 *Factor Graphs*

Factor graphs (Loeliger, 2004) already introduced in background Section 2.4.1, are a convenient way of representing graphical models. To recall the definition, a factor graphs is a bipartite graph that interconnects a set of factors $f_s(\mathbf{x}_s)$ with

a set of variables $\mathbf{x} = \{x_1, \ldots, x_i\}$, each factor defining dependencies among its subset of variables. A global probability distribution $p(\mathbf{x})$ can be defined as the product of all these factors $p(\mathbf{x}) = \frac{1}{Z} \prod_{s \in \mathcal{F}} f_s(\mathbf{x}_s)$, where Z is the normalization constant of the probability distribution. A visual representation of a Factor Graph is depicted in the left image of Figure 7.

4.2.2 *Belief Propagation*

Belief Propagation (Bishop, 2006), also known as the sum-product algorithm, is a message passing algorithm that performs inference on graphical models by locally marginalizing over random variables. It exploits the structure of factor graphs, allowing more efficient computation of the marginals. Belief Propagation directly operates on factor graphs by sending messages (real valued functions) on its edges. These messages exchange 'beliefs' of the sender nodes about the receiver nodes, thereby transporting information about the variable's probabilities. We can distinguish two types of messages: those that go from variables to factors and those that go from factors to variables.

Variable to factor: $\mu_{x_m \to f_s}(x_m)$ is the product of all incoming messages to variable x_m from all neighbor factors $\mathcal{N}(x_m)$ except for factor f_s.

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in \mathcal{N}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m) \tag{30}$$

Factor to variable: $\mu_{f_s \to x_i}(x_i)$ is the product of the factor f_s itself with all its incoming messages from all variable neighbor nodes except for x_i marginalized over all associated variables $\mathbf{x}_s$ except x_i.

$$\mu_{f_s \to x_i}(x_i) = \sum_{\mathbf{x}_s \setminus x_i} f_s(\mathbf{x}_s) \prod_{m \in \mathcal{N}(f_s) \setminus n} \mu_{x_m \to f_s}(x_m) \tag{31}$$

To run the Belief Propagation algorithm, messages are initialized with uniform probabilities, and the two above mentioned operations are then recursively run until convergence. One can subsequently obtain marginal estimates $p(x_i)$ by multiplying all incoming messages from the neighboring factors:

$$p(x_i) \propto \prod_{s \in \mathcal{N}(x_i)} \mu_{f_s \to x_i}(x_i) \tag{32}$$

From now on, we simplify notation by removing the argument of the messages function, such that the variable to factor messages simplify to $\mu_{f_s \to x_i}$ and the factor to variable messages simplify to $\mu_{f_s \to x_i}$. When we want to refer to all set of messages in the graph from a factors to variables instead of to single one, we denote it $\mu_{f \to x}$ and the set of messages in the graph from variables to factors $\mu_{x \to f}$. In the left side of Figure 7 we can see the defined messages on a factor graph where black squares represent factors and blue circles represent variables.

4.2.3 *LDPC Codes*

In this chapter we will apply our proposed method to error correction decoding. Low Density Parity Check (LDPC) codes (Gallager, 1962; MacKay, 2003) are linear codes used to correct errors in data transmitted through noisy communication channels. The sender encodes the data with redundant bits while the receiver has to decode the original message. In an LDPC code, a parity check sparse matrix $\mathbf{H} \in \mathbb{B}^{(n-k)\times n}$ is designed, such that given a code-word $\mathbf{x} \in \mathbb{B}^n$ of n bits the product of $\mathbf{H}$ and $\mathbf{x}$ is constrained to equal zero: $\mathbf{Hx} = \mathbf{0}$. $\mathbf{H}$ can be interpreted as an adjacency matrix that connects n variables (i.e. the transmitted bits) with $(n-k)$ factors, i.e. the parity checks that must sum to 0. The entries of $\mathbf{H}$ are 1 if there is an edge between a factor and a variable, where rows index factors and columns index variables. For a linear code (n, k), n is the number of variables and $(n-k)$ the number of factors. The prior probability of the transmitted code-word $P(\mathbf{x}) \propto \mathbb{1}[\mathbf{Hx} = \mathbf{0} \bmod 2]$ can be factorized as:

$$P(\mathbf{x}) \propto \prod_s \mathbb{1}\Big[\sum_{n \in \mathcal{N}(s)} x_n = 0 \bmod 2\Big] = \prod_s f_s(\mathbf{x}_s) \tag{33}$$

At the receiver we get a noisy version of the code-word, $\mathbf{r}$. The noise is assumed to be i.i.d, therefore we can express the probability distribution of the received code-word as $\mathbf{x}$ as $P(\mathbf{r}|\mathbf{x}) = \prod_n P(r_n|x_n)$. Finally we can express the posterior distribution of the transmitted code-word given the received signal as:

$$P(\mathbf{x}|\mathbf{r}) \propto P(\mathbf{x})P(\mathbf{r}|\mathbf{x}) \tag{34}$$

Equation 34 is a product of factors, where some factors in $P(\mathbf{x})$ (eq. 33) are connected to multiple variables expressing a constraint among them. Other factors $P(\mathbf{r}|\mathbf{x})$ are connected to a single variable expressing a prior distribution for that variable. A visual representation of this factor graph is shown in the left image of Figure 7. Finally, in order to infer the transmitted code-word $\mathbf{x}$ given $\mathbf{r}$, we can just run (loopy) Belief Propagation described in section 4.2.2 on the Factor Graph described above (equation 34). In other words, error correction with LDPC codes can be interpreted as an instance of Belief Propagation applied to its associated factor graph.

4.3 RELATED WORK

One of the closest works to this method is (Satorras et al., 2019) also mentioned in the previous Chapter, which also combines graphical inference with graph neural networks. However, the previous chapter model is only applied to the Kalman Filter, a Hidden Gaussian Markov model for time sequences, and all factor graphs are assumed to be pair-wise. In this new chapter, we run the GNN in arbitrary Factor Graphs, and we hybridize Belief Propagation, which allows us to enhance one of its main applications (LDPC codes). Other works from literature also learn an inference model from data like Recurrent Inference Machines (Putzky and Welling, 2017) and Iterative Amortized Inference (Marino et al., 2018). However, in our case we are presenting a hybrid algorithm instead of a fully learned one. Additionally in (Putzky and Welling, 2017) graphical models play no role.

Our work is also related to meta learning (Schmidhuber, 1987; Andrychowicz et al., 2016) in the sense that it learns a more flexible algorithm on top of an already existing one. It also has some interesting connections to the ideas from the consciousness prior (Bengio, 2017) since our model is an actual implementation of a sparse factor graph that encodes prior knowledge about the task to solve.

Another interesting line of research concerns the convergence of graphical models with neural networks. In (Mirowski and LeCun, 2009), the conditional probability distributions of a graphical model are replaced with trainable factors. (Johnson et al., 2016) learns a graphical latent representation and runs Belief Propagation on it. Combining the strengths of convolutional neural networks and conditional random fields has shown to be effective in image segmentation tasks (Chen et al., 2014; Zheng et al., 2015). A model to run Neural Networks on factor graphs was also introduced in (Zhang et al., 2019). However, in our case, we simply adjust the Graph Neural Network equations to the factor graph scenario as a building block for our hybrid model (NEBP).

More closely to our work, (Yoon et al., 2018) trains a graph neural network to estimate the marginals in Binary Markov Random Fields (Ising model) and the performance is compared with Belief Propagation for loopy graphs. In our work we are proposing a hybrid method that combines the benefits of both GNNs and BP in a single model. In (Nachmani et al., 2016) some weights are learned in the edges of the Tanner graph for High Density Parity Check codes, in our case we use a GNN on the defined graphical model and we test our model on Low Density Parity Check codes, one of the standards in communications for error decoding. A subsequent work (Liu and Poulin, 2019) uses the model from (Nachmani et al., 2016) for quantum error correcting codes. Recently, (Kuck et al., 2020) presented a strict generalization of Belief Propagation with Neural Networks, in contrast, our model augments Belief Propagation with a Graph Neural Network which learned messages are not constrained to the message passing scheme of

Belief Propagation, refining BP messages without need to backpropagate through them.

4.4 METHOD

4.4.1 *Graph Neural Network for Factor Graphs*

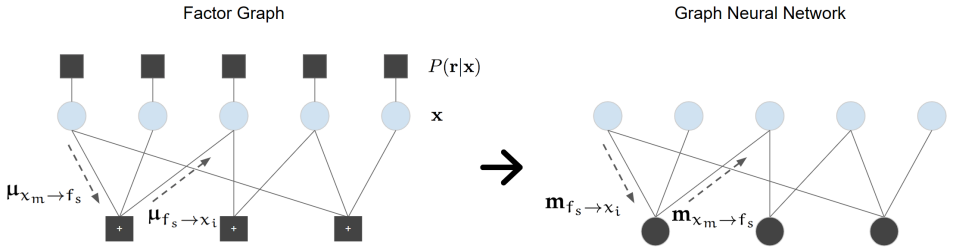

Figure 7: Visual representation of a LDPC Factor Graph (left) and its equivalent representation in our Graph Neural Network (right). In the Factor Graph, factors are displayed as black squares, variables as blue circles. In the Graph Neural Network, nodes associated to factors are displayed as black circles. Nodes associated to variables are displayed as blue circles.

We will propose a hybrid method to improve Belief Propagation (BP) by combining it with Graph Neural Networks (GNNs). Both methods can be seen as message passing on a graph. However, where BP sends messages that follow directly from the definition of the graphical model, messages sent by GNNs must be learned from data. To achieve seamless integration of the two message passing algorithms, we will first extend GNNs to factor graphs.

Graph Neural Networks (Bruna et al., 2013; Defferrard et al., 2016; Kipf and Welling, 2016a) operate on graph-structured data by modelling interactions between pairs of nodes. A graph is defined as a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with nodes $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$. For a recap, a more precise definition of Graph Neural Network and the detailed equations are provided in the Background Section 2.3.

In order to integrate the GNN messages with those of BP we have to run them on a Factor Graph. In (Yoon et al., 2018) a GNN was defined on pair-wise factor graphs (ie. a factor graph where each factor contains only two variables). In their work each variable of the factor graph represents a node in the GNN, and each factor connecting two variables represented an edge in the GNN. Properties of the factors were associated with edge attributes $\mathbf{a}_{ij}$. The mapping between GNNs and Factor Graphs becomes less trivial when each factor may contain more than two variables. We can then no longer consider each factor as an edge of the GNN. In this work we propose special case of GNNs to run on factor graphs with an arbitrary number of variables per factor.

Similarly to Belief Propagation, we first consider a Factor Graph as a bipartite graph $\mathcal{G}_f = (\mathcal{V}_f, \mathcal{E}_f)$ with two type of nodes $\mathcal{V}_f = \mathcal{X} \cup \mathcal{F}$, variable-nodes $v_x \in \mathcal{X}$ and factor-nodes $v_f \in \mathcal{F}$, and two types of edge interactions, depending on if they go from factor-node to variable-node or vice-versa. With this graph definition, all interactions are again pair-wise (between factor-nodes and variable-nodes in the bipartite graph).

Figure 7. illustrates how a factor graph is transformed into its equivalent representation for our GNN. Each variable in the factor graph, depicted by blue circles, corresponds to a node in the GNN. However, Factors (shown as black squares in the factor graph) are only represented in the GNN if they connect to multiple variables; single edge factors are treated as attributes of their connected variable-node, eliminating unnecessary nodes. Once we have defined our graph, we use the GNN notation from the Background Section 2.3, and we adapt it specifically for this factor graph structre. From now on we reference the new GNN for factor graphs as FG-GNN. Within this framework, we introduce the term FG-GCL to describe the Graph Convolutional Layer for factor graphs, which define the layers of the FG-GNN. The definition of a FG-GCL is detailed in Table 4.

Table 4: Graph Convolutional Layer of a FG-GNN.

FG-GCL	
Edge message	$\mathbf{m}_{x_m \to f_s} = \phi_{x \to f}(\mathbf{h}^l_{f_s}, \mathbf{h}^l_{x_m}, \mathbf{a}_{x_m \to f_s})$ $\mathbf{m}_{f_s \to x_i} = \phi_{f \to x}(\mathbf{h}^l_{x_i}, \mathbf{h}^l_{f_s}, \mathbf{a}_{f_s \to x_i})$
Aggregation	$\mathbf{m}_{f_s} = \sum_{x_m \in \mathcal{N}(f_s)} \mathbf{m}_{x_m \to f_s}$ $\mathbf{m}_{x_i} = \sum_{f_s \in \mathcal{N}(x_i)} \mathbf{m}_{f_s \to x_i}$
Node Update	$\mathbf{h}^{l+1}_{f_s} = \phi_{v_f}(\mathbf{h}^l_{f_s}, \mathbf{m}_{f_s}, \mathbf{a}_{f_s})$ $\mathbf{h}^{l+1}_{x_i} = \phi_{v_x}(\mathbf{h}^l_{x_i}, \mathbf{m}_{x_i}, \mathbf{a}_{x_i})$

In the GNN we did not have two different kind of variables in the graph and hence we only used one edge function ϕ_e. In the FG-GNN however, we now have two types of nodes, which lead to to two type of edges which we model with different functions $\phi_{x \to f}$ and $\phi_{f \to x}$, depending on whether the message was sent by a variable or a factor node. In addition, we also have two type of node embeddings $\mathbf{h}_{x_i}$ and $\mathbf{h}_{f_s}$ for the two types of nodes v_x and v_f. Again we sum over all incoming messages for each node, but now in the node update we have two different functions, ϕ_{v_f} for the factor-nodes and ϕ_{v_x} for the variable-nodes. The optional edge attributes are now named $\mathbf{a}_{x_i \to f_s}$ and $\mathbf{a}_{f_s \to x_i}$. The node attributes are named $\mathbf{a}_{f_s}$ and $\mathbf{a}_{x_i}$.

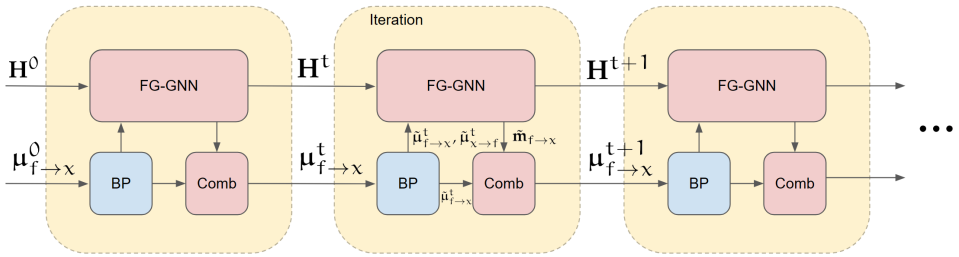

Figure 8: Graphical illustration of our Neural Enhanced Belief Propagation algorithm. Three modules are depicted in each iteration {BP, FG-GNN, Comb.}. Each module is associated to each one of the three lines from Equation 35.

4.4.2 *Notation Clarification*

Differently to the standard notation described in Section 2.1, we refer to the embedding variables of the whole graph by omitting the subindex instead of using a capital letter. For example, if $\mathbf{h}_{x_i} \in \mathrm{R}^{nf}$ is the embedding of a variable node x_i with "nf" being the number of features in the embedding, then we refer to the set of all variable node embeddings as $\mathbf{h}_x$. We follow this convention for the rest of embeddings in the neural network, such that $\mathbf{h}_f$, $\mathbf{m}_{x\to f}$ represent the set of embeddings in the graph, with their respective node-wise and edge-wise embeddings being $\mathbf{h}_{f_s}$, $\mathbf{m}_{f_s\to x_i}$.

4.4.3 *Neural Enhanced Belief Propagation*

Now that we have defined the FG-GNN we can introduce our hybrid method that runs co-jointly with Belief Propagation on a factor graph, we denote this new method Neural Enhanced Belief Propagation (NEBP). At a high level, the procedure is as follows: after every Belief Propagation iteration, we input the BP messages into the FG-GNN. The FG-GNN then runs for two iterations and updates the BP messages. This step is repeated recursively for N iterations. After that, we can compute the marginals from the refined BP messages.

We first define the two functions BP$(\cdot)$ for Belief Propagation and FG-GNN$(\cdot)$. BP$(\cdot)$ takes as input all the factor-to-node messages $\mu^t_{f\to x}$, then runs the two BP updates eqns. 30 and 31 respectively and outputs the result of that computation as $\tilde{\mu}^t_{f\to x}, \tilde{\mu}^t_{x\to f}$. We initialize $\mu^0_{f\to x}$ as uniform distributions.

The function FG-GNN$(\cdot)$ runs the equations displayed in Table 4. At every "t" iteration we give it as input the quantities $\mathbf{H}^t = \{\mathbf{h}^t_x \cup \mathbf{h}^t_f\}$, $\mathbf{a}_{x\to f}$, $\mathbf{a}_{f\to x}$ and $\mathbf{a}_v$. $\mathbf{H}^t$ is initialized randomly at $\mathbf{H}^0$ by sampling from a normal distribution. Notice this $\mathbf{H}$ is a different variable than $\mathbf{H}$ from Section 4.2.3. Moreover, the attributes $\mathbf{a}_{x\to f}$ and $\mathbf{a}_{f\to x}$ are provided to the function FG-GNN$(\cdot)$ as the messages $\tilde{\mu}^t_{x\to f}$

and $\tilde{\mu}^t_{f\to x}$ obtained from BP$(\cdot)$, as an exception, the subset of messages from $\tilde{\mu}_{f\to x}$ that go from a singleton factor f_j to its neighbor variable are treated as attributes $\mathbf{a}_v$. The output of the FG-GNN$(\cdot)$ are the updated latent node representations $\mathbf{H}^{t+1}$ and the latent messages representations $\tilde{\mathbf{m}}_{f\to x}$, which are computed as part of the FG-GNN algorithm in Table 4. These latent representations $\tilde{\mathbf{m}}_{f\to x} = \mathbf{m}_{f\to x} \cup \mathbf{h}^{t+1}_x$ will be used to update the current message estimates $\tilde{\mu}^t_{f\to x}$, specifically $\mathbf{h}^{t+1}_x$ will refine those messages $\tilde{\mu}^t_{f_j\to x}$ that go from a singleton factor to its neighbor variable, and $\mathbf{m}_{f\to x}$ will refine the rest of messages $\tilde{\mu}^t_{f\to x}\backslash\tilde{\mu}^t_{f_j\to x}$. All other variables computed inside FG-GNN$(\cdot)$ are kept internal to this function.

Finally, $f_s(\cdot)$ and $f_u(\cdot)$ take as input the embeddings $\mathbf{m}_{f\to x}$ and output a refinement for the current message estimates $\tilde{\mu}^t_{f\to x}$. Particularly, $f_s(\cdot)$ outputs a positive scalar value that multiplies the current estimate, and $f_u(\cdot)$ outputs a positive vector which is summed to the estimate. Both functions encompass two Multi Layer Perceptrons (MLP), one MLP takes as input the node embeddings $\mathbf{h}^{t+1}_x$, and outputs the refinement for the singleton factor messages, the second MLP takes as input the edge embeddings $\mathbf{m}^t_{f\to x}$ and outputs a refinement for the rest of messages $\tilde{\mu}^t_{f\to x}\backslash\tilde{\mu}^t_{f_l\to x}$. In summary, the hybrid algorithm thus looks as follows:

$$\begin{aligned} \tilde{\mu}^t_{f\to x}, \tilde{\mu}^t_{x\to f} &= \text{BP}\big(\mu^t_{f\to x}\big) \\ \mathbf{H}^{t+1}, \tilde{\mathbf{m}}_{f\to x} &= \text{FG-GNN}\big(\mathbf{H}^t, \tilde{\mu}^t_{f\to x}, \tilde{\mu}^t_{x\to f}\big) \\ \mu^{t+1}_{f\to x} &= \tilde{\mu}^t_{f\to x} f_s(\tilde{\mathbf{m}}_{f\to x}) + f_u(\tilde{\mathbf{m}}_{f\to x}) \end{aligned} \tag{35}$$

After running the algorithm for N iterations. We obtain the estimate $\hat{p}(x_i)$ by using the same operation as in Belief Propagation (eq. 32), which amounts to taking a product of all incoming messages to node x_i, i.e. $\hat{p}(x_i) \propto \prod_{s\in\mathcal{N}(x_i)} \mu_{f_s\to x_i}$. From these marginal distributions we can compute any desired quantity on a node.

4.4.4 *Training and Loss*

The loss is computed from the estimated marginals $\hat{p}(\mathbf{x})$ and ground truth values $\mathbf{x}_{gt}$, which we assume known during training. In the LDPC experiment the ground truths $\mathbf{x}_{gt}$ are the transmitted bits which are known by the receiver during the training stage.

$$\text{Loss}(\Theta) = \mathcal{L}\left(\mathbf{x}_{gt}, \hat{p}(\mathbf{x})\right) + \mathcal{R} \tag{36}$$

During training we back-propagate through the whole multi-layer estimation model (with each layer an iteration of the hybrid model), updating the FG-GNN, $f_s(\cdot)$ and $f_u(\cdot)$ weights Θ. The number of training iterations is chosen by cross-validating. In our experiments we use the binary cross entropy loss for $\mathcal{L}$. The reg-

ularization term $\mathcal{R}$ is the mean of $f_u(\cdot)$ outputs, i.e. $\mathcal{R} = \frac{1}{N}\sum_t \text{mean}\big(f_u(\tilde{\mathbf{m}}^t_{f\to x})\big)$. It encourages the model to behave closer to Belief Propagation. In case, $f_u(\cdot)$ output is set to 0, the hybrid algorithm would be equivalent to Belief Propagation. This happens because $f_s(\cdot)$ outputs a scalar value that on its own only modifies the norm of $\tilde{\mu}^t_{f\to x}$. Belief Propagation can operate on unnormalized beliefs, although it is a common practice to normalize them at every BP$(\cdot)$ iteration to avoid numerical instabilities. Therefore, only modifying the norm of the messages on its own doesn't change our hybrid algorithm because messages are being normalized at every BP iteration.

4.5 EXPERIMENTS

We analyze the performance of Belief Propagation, FG-GNNs, and our Neural Enhanced Belief Propagation (NEBP) in an error correction task where Belief Propagation is also known as LDPC (Section 4.2.3). In both FG-GNN and NEBP, the edge operations $\sigma_{x\to f}$, $\sigma_{f\to x}$ defined in Section 4.4 consist of two layers Multilayer Perceptrons (MLP). The node update functions σ_{v_f} and σ_{v_x} consist of two layer MLPs followed by a Gated Recurrent Unit (Chung et al., 2014). Functions $f_s(\cdot)$ and $f_u(\cdot)$ from the NEBP combination module also contain two layers MLPs.

4.5.1 *Low Density Parity Check Codes*

LDPC codes, explained in Section 4.2.3 are a particular case of Belief Propagation run on a bipartite graph for error correction decoding tasks. Bipartite graphs contain cycles, hence Belief Propagation is no longer guaranteed to converge nor to provide the optimal estimates. Despite this lack of guarantees, LDPC has shown excellent results near the Shannon limit (MacKay and Neal, 1996) for Gaussian channels. LDPC assumes a channel with an analytical solution, commonly a Gaussian channel. In real world scenarios, the channel may differ from Gaussian or it may not even have a clean analytical solution to run Belief Propagation on, leading to sub-optimal estimates. An advantage of neural networks is that, in such cases, they can learn a decoding algorithm from data.

In this experiment we consider the bursty noisy channel from (Kim et al., 2018), where a signal x_i is transmitted through a standard Gaussian channel $z_i \sim \mathcal{N}(0, \sigma_c^2)$, however this time, a larger noise signal $w_i \sim \mathcal{N}(0, \sigma_b^2)$ is added with a small probability ρ. More formally:

$$r_i = x_i + z_i + p_i w_i \tag{37}$$

Where r_i is the received signal, and p_i follows a Bernoulli distribution such that $p_i = 1$ with probability ρ, and $p_i = 0$ with probability $1 - \rho$. In our experiments, we set $\rho = 0.05$ as done in (Kim et al., 2018). This bursty channel describes how unexpected signals may cause interference in the middle of a transmitted frame.

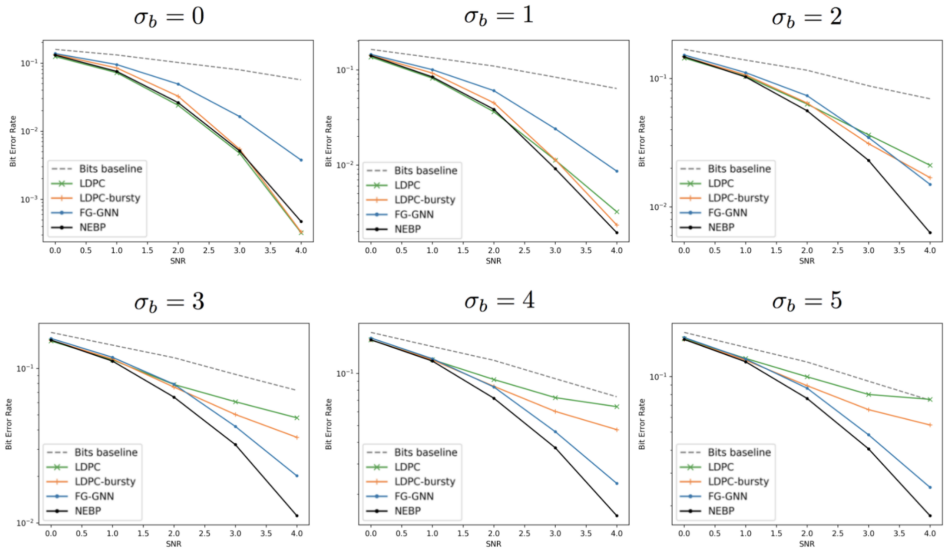

Figure 9: Bit Error Rate (BER) with respect to the Signal to Noise Ratio (SNR) for different bursty noise values $\sigma_b \in \{0, 1, 2, 3, 4, 5\}$.

For example, radars may cause bursty interference in wireless communications. In LDPC, the SNR is assumed to be known and fixed for a given frame, yet, in practice it needs to be estimated with a known preamble (the pilot sequence) transmitted before the frame. If bursty noise occurs in the middle of the transmission, the estimated SNR is blind to this new noise level.

DATASET: We use the parity check matrix $\mathbf{H}$ "96.3.963" from (MacKay and Codes, 2009) for all experiments, with $n = 96$ variables and $k = 48$ factors, i.e. a transmitted code-word $\mathbf{x} \in \mathbb{B}^n$ contains 96 bits. The training dataset consists of pairs of received and transmitted code-words $\{(\mathbf{r}_d, \mathbf{x}_d)\}_{1 \leqslant d \leqslant L}$. The transmitted code-words $\mathbf{x}$ are used as ground truth for training the decoding algorithm. The received code-words $\mathbf{r}$ are obtained by transmitting $\mathbf{x}$ through the bursty channel from Equation 37. We generate samples for $\text{SNR}_{db} = \{0, 1, 2, 3, 4\}$. Regarding the bursty noise σ_b, we randomly sample its standard deviation from a uniform distribution $\sigma_b \in [0, 5]$. We generate a validation partition of 750 code-words (150 code-words per SNR_{db} value). For the training partition we keep generating samples until convergence, i.e. until we do not see further improvement in the validation accuracy.

TRAINING PROCEDURE: We provide as input to the model the received code-word $\mathbf{r}_d$ and the SNR for that code-word. These values are provided as node attributes a_v described in Section 4.4. We run the algorithms for 20 iterations and the loss is computed as the cross entropy between the estimated $\hat{\mathbf{x}}$ and the ground truth $\mathbf{x}_d$. We use an Adam optimizer (Kingma and Ba, 2014) with a learning rate

$2e^{-4}$ and batch size of 1. The number of hidden features is 32 and all activation functions are 'Selus' (Klambauer et al., 2017). As a evaluation metric we compute the Bit Error Rate (BER), which is the number of error bits divided by the total amount of transmitted bits. The number of test code-words we used to evaluate each point from our plots (Figure 9) is at least $\frac{200}{\hat{\text{BER}} \cdot n}$, where n is the number of bits per code-word and $\hat{\text{BER}}$ is the estimated Bit Error Rate for that point.

BASELINES: Beside the already mentioned methods (FG-GNN and standard LDPC error correction decoding), we also run two extra baselines. The first one we call *Bits baseline*, which consists of independently estimating each bit that maximizes $p(r_i|x_i)$. The other baseline, called *LDPC-bursty*, is a variation of LDPC, where instead of considering a SNR with a noise level $\sigma_c^2 = \text{var}[z]$, we consider the noise distribution from Equation 37 such that now the noise variance is $\sigma^2 = \text{var}[z + pw] = \sigma_c^2 + (\rho(1-\rho) + \rho^2)\mathbb{E}_{\sigma_b^2}[\sigma_b^2]$. This is a fairer comparison to our learned methods, because even if we are blind to the σ_b value, we know there may be a bursty noise with probability ρ and $\sigma_b \sim \mathcal{U}(0, 5)$.

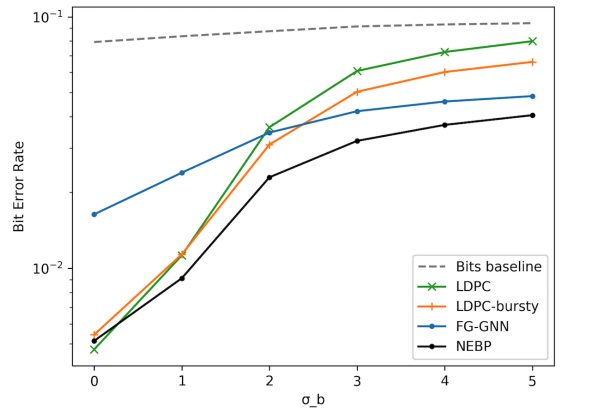

Figure 10: Bit Error Rate (BER) with respect to σ_b value for a fixed SNR=3.

RESULTS: In Figure 9 we show six different plots for each of the σ_b values {0, 1, 2, 3, 4, 5}. In each plot we sweep the SNR from 0 to 4. Notice that for $\sigma_b = 0$ the bursty noise is non-existent and the channel is equivalent to an Additive White Gaussian Noise channel (AWGN). LDPC has analytically been designed for this channel obtaining its best performance here. The aim of our algorithm is to outperform LDPC for $\sigma_b > 0$ while still matching its performance for $\sigma_b = 0$. As shown in the plots, as σ_b increases, the performance of NEBP and FG-GNN improves compared to the other methods, with NEBP always achieving the best performance, and getting close to the LDPC performance for the AWGN channel ($\sigma_b = 0$). In summary, the hybrid method is more robust than LDPC, obtaining competitive results to LDPC for AWGN channels but still outperforming it when

bursty interferences are present. The FG-GNN instead, obtains relatively poor performance compared to LDPC for small σ_b, demonstrating that belief propagation is still a very powerful tool compared to pure learned inference for this task. Our NEBP is able to combine the benefits from both LDPC and the FG-GNN to achieve the best performance, exploiting the adaptability of FG-GNN and the prior knowledge of Belief Propagation. Finally, LDPC-bursty shows a more robust performance as we increase σ_b but it is significantly outperformed by NEBP in bursty channels, and it also performs slightly worse than LDPC for the AWGN channel ($\sigma_b = 0$).

In order to better visualize the decrease in performance as the burst variance increases, we sweep over different σ_b values for a fixed SNR=3. The result is shown in Figure 10. The larger σ_b, the larger the BER. However, the performance decreases much less for our NEBP method than for LDPC and LDPC-bursty. In other words, NEBP is more robust as we move away from the AWGN assumption. We want to emphasize that in real world scenarios, the channel may always deviate from gaussian. Even if assuming an AWGN channel, its parameters (SNR) must be estimated in real scenarios. This potential deviations make hybrid methods a very promising approach.

4.6 CONCLUSIONS

In this chapter, we presented a hybrid inference method that enhances Belief Propagation by co-jointly running a Graph Neural Network that we designed for factor graphs. In cases where the data generating process is not fully known (e.g. the parameters of the graphical model need to be estimated from data), belief propagation doesn't perform optimally. Our hybrid model in contrast is able to combine the prior knowledge encoded in the graphical model (albeit with the wrong parameters) and combine this with a (factor) graph neural network with its parameters learned from labeled data on a representative distribution of channels. Note that we can think of this as meta-learning because the FG-GNN is not trained on one specific channel but on a distribution of channels and therefore must perform well on any channel sampled from this distribution without knowing its specific parameters. We tested our ideas on a state-of-the-art LDPC implementation with realistic bursty noise distributions. Our experiments show that the neural enhancement of LDPC improves performance both relative to LDPC and relative to FG-GNN as the variance in the bursts gets larger.

Part II

E(N) EQUIVARIANT GRAPH NEURAL NETWORKS

5

E(N) EQUIVARIANT GRAPH NEURAL NETWORKS

This Chapter is based on the content of:
Victor Garcia Satorras et al. (2021b). "E (n) equivariant graph neural networks." In: *International conference on machine learning*. PMLR, pp. 9323–9332

5.1 INTRODUCTION

In the following chapters we focus on developing machine learning models that leverage symmetries inherent in the data. Specifically, we focus on the Euclidean symmetry group which includes translations, rotations and reflections. In this chapter we design a variant of graph neural networks that is equivariant to Euclidean transformations which we name E(n) Equivariant Graph Neural Networks (EGNNs).

Although deep learning has largely replaced hand-crafted features, many advances are critically dependent on inductive biases in deep neural networks. An effective method to restrict neural networks to relevant functions is to exploit the *symmetry* of problems by enforcing equivariance with respect to transformations from a certain symmetry group. Notable examples are translation equivariance in Convolutional Neural Networks and permutation equivariance in Graph Neural Networks (Bruna et al., 2013; Defferrard et al., 2016; Kipf and Welling, 2016a).

Many problems exhibit 3D translation and rotation symmetries. Some examples are point clouds (Uy et al., 2019), 3D molecular structures (Ramakrishnan et al., 2014) or N-body particle simulations (Kipf et al., 2018). The group corresponding to these symmetries is named the Euclidean group: SE(3) or when reflections are included E(3). It is often desired that predictions on these tasks are either equivariant or invariant with respect to E(3) transformations.

Recently, various forms and methods to achieve E(3) or SE(3) equivariance have been proposed (Thomas et al., 2018b; Fuchs et al., 2020; Finzi et al., 2020b; Köhler et al., 2020a). Many of these works achieve innovations in studying types of

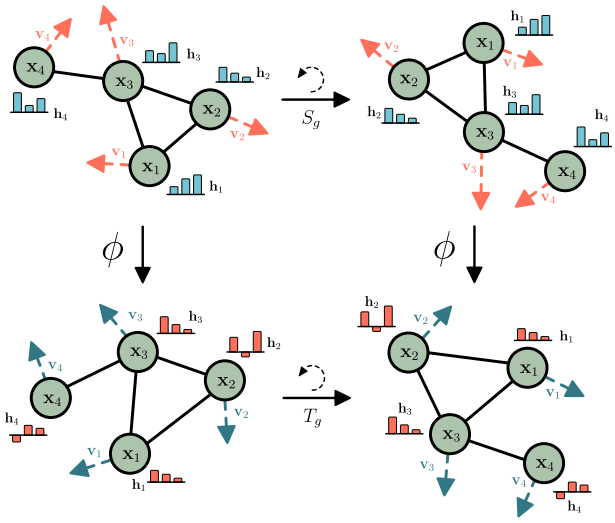

Figure 11: Example of rotation equivariance on a graph with an equivariant graph neural network ϕ.

higher-order representations for intermediate network layers. However, the transformations for these higher-order representations require coefficients or approximations that can be expensive to compute. Additionally, in practice for many types of data the inputs and outputs are restricted to scalar values (for instance temperature or energy, referred to as type-0 in literature) and 3d vectors (for instance velocity or momentum, referred to as type-1 in literature).

In this work we present a new architecture that is translation, rotation and reflection equivariant ($E(n)$), and permutation equivariant with respect to an input set of points. Our model is simpler than previous methods in that it does not require the spherical harmonics as in (Thomas et al., 2018b; Fuchs et al., 2020) while it can still achieve competitive or better results. In addition, equivariance in our model is not limited to the 3-dimensional space and can be scaled to larger dimensional spaces without a significant increase in computation.

We evaluate our method in modelling dynamical systems, representation learning in graph autoencoders and predicting molecular properties in the QM9 dataset. Our method reports the best or very competitive performance in all three experiments.

5.2 BACKGROUND

In this section we introduce the relevant materials on equivariance and graph neural networks which will later complement the definition of our method.

5.2.1 *Equivariance*

The concept of equivariance is introduced in more detail in the Background Section 2.5. For a quick recapitulation, consider a set of M points in an n-dimensional space $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_M\} \in \mathbb{R}^{M \times n}$. Let $T_g : X \to X$ be a set of transformations on X for the abstract group $g \in G$. We say a function $\phi : X \to Y$ is equivariant to g if there exists an equivalent transformation on its output space $S_g : Y \to Y$ such that:

$$\phi(T_g(\mathbf{X})) = S_g(\phi(\mathbf{X})) \tag{38}$$

In this particular chapter we are interested in equivariance to Euclidean transformations, such as translations, rotations, and reflections. As well as to permutations, which are already inherent in graph neural networks. Our method introduced in this chapter will satisfy these four mentioned equivariant constraints.

5.2.2 *Graph Neural Networks*

Graph Neural Networks are permutation equivariant networks that operate on graph structured data (Bruna et al., 2013; Defferrard et al., 2016; Kipf and Welling, 2016a). They have been introduced in more detail in the Background Section 2.3. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $v_i \in \mathcal{V}$ and edges $e_{ij} \in \mathcal{E}$ we can recall the equations of a graph convolutional layer from Section 2.3 as:

$$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{a}_{ij}) \tag{39}$$

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij} \tag{40}$$

$$\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i) \tag{41}$$

Where $\mathbf{h}_i^l \in \mathbb{R}^{nf}$ is the nf-dimensional embedding of node v_i at layer l. $\mathbf{a}_{ij}$ are the edge attributes. $\mathcal{N}(i)$ represents the set of neighbors of node v_i. Finally, ϕ_e and ϕ_h are the edge and node operations respectively which are commonly approximated by Multilayer Perceptrons (MLPs). The E(n) Equivariant Graph Neural Network introduced in this chapter is build on top this Graph Neural Network equations.

5.3 EQUIVARIANT GRAPH NEURAL NETWORKS

In this section we present Equivariant Graph Neural Networks (EGNNs). Following the notation from background Section 5.2.2, we consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $v_i \in \mathcal{V}$ and edges $e_{ij} \in \mathcal{E}$. In addition to the feature node embeddings $\mathbf{h}_i \in \mathbb{R}^{nf}$ we now also consider a n-dimensional coordinate $\mathbf{x}_i \in \mathbb{R}^n$ associated

with each of the graph nodes. Our model will preserve equivariance to rotations and translations on these set of coordinates $\mathbf{x}_i$ and it will also preserve equivariance to permutations on the set of nodes $\mathcal{V}$ in the same fashion as GNNs.

Our Equivariant Graph Convolutional Layer (EGCL) takes as input the set of node embeddings $\mathbf{H}^l = \{\mathbf{h}_0^l, \dots, \mathbf{h}_{M-1}^l\}$, coordinates $\mathbf{X}^l = \{\mathbf{x}_0^l, \dots, \mathbf{x}_{M-1}^l\}$ and edge information $\mathcal{E} = (e_{ij})$ and outputs a transformation on $\mathbf{H}^{l+1}$ and $\mathbf{X}^{l+1}$. Concisely: $\mathbf{H}^{l+1}, \mathbf{X}^{l+1} = \text{EGCL}[\mathbf{H}^l, \mathbf{X}^l, \mathcal{E}]$. The equations that define this layer are the following:

$$\mathbf{m}_{ij} = \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2, \mathbf{a}_{ij}\right) \tag{42}$$

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \tag{43}$$

$$\mathbf{m}_i = \sum_{j\neq i}\mathbf{m}_{ij} \tag{44}$$

$$\mathbf{h}_i^{l+1} = \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right) \tag{45}$$

Notice the main differences between the above proposed method and the original Graph Neural Network from Equation 39 are found in equations 42 and 43. In equation 42 we now input the relative squared distance between two coordinates $\|\mathbf{x}_i^l - \mathbf{x}_j^l\|^2$ into the edge operation ϕ_e. The embeddings $\mathbf{h}_i^l$, $\mathbf{h}_j^l$, and the edge attributes a_{ij} are also provided as input to the edge operation as in the GNN case. In our case the edge attributes will incorporate the edge values $\mathbf{a}_{ij} := e_{ij}$, but they can also include additional edge information.

In Equation 43 we update the position of each particle $\mathbf{x}_i$ as a vector field in a radial direction. In other words, the position of each particle $\mathbf{x}_i$ is updated by the weighted sum of all relative differences $(\mathbf{x}_i - \mathbf{x}_j)_{\forall j}$. The weights of this sum are provided as the output of the function $\phi_x : \mathbb{R}^{nf} \to \mathbb{R}^1$ that takes as input the edge embedding $\mathbf{m}_{ij}$ from the previous edge operation and outputs a scalar value. C is chosen to be $1/(M-1)$, which divides the sum by its number of elements. This equation is the main difference of our model compared to standard GNNs and it is the reason why equivariances 1, 2 are preserved (proof in Appendix B.1). Despite its simplicity, this equivariant operation is very flexible since the embedding $\mathbf{m}_{ij}$ can carry information from the whole graph and not only from the given edge e_{ij}.

Finally, equations 44 and 45 follow the same updates than standard GNNs. Equation 44 is the aggregation step, in this work we choose to aggregate messages from all other nodes $j \neq i$, but we could limit the message exchange to a given neighborhood $j \in \mathcal{N}(i)$ if desired in both equations 44 and 43. Equation 45 per-

forms the node operation ϕ_h which takes as input the aggregated messages $\mathbf{m}_i$, the node emedding $\mathbf{h}_i^l$ and outputs the updated node embedding $\mathbf{h}_i^{l+1}$.

5.3.1 *Analysis on E(n) Equivariance*

In this section we analyze the equivariance properties of our model for E(3) symmetries (i.e. properties 1 and 2 stated in section 2.5). In other words, our model should be translation equivariant on $\mathbf{X}$ for any translation vector $g \in \mathbb{R}^n$ and it should also be rotation and reflection equivariant on $\mathbf{X}$ for any orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$. Let $\mathbf{QX}$ be shorthand for $(\mathbf{Qx}_1, \ldots, \mathbf{Qx}_M)$, more formally our model satisfies:

$$\mathbf{QX}^{l+1} + \mathbf{g}, \mathbf{H}^{l+1} = \text{EGCL}(\mathbf{QX}^l + \mathbf{g}, \mathbf{H}^l)$$

We provide a formal proof of this in Appendix B.1. Intuitively, let's consider a $\mathbf{h}_i^l \in \mathbf{H}^l$ feature which is already E(n) invariant, then we can see that the resultant edge embedding $\mathbf{m}_{ij}$ from Equation 42 will also be E(n) invariant, because in addition to $\mathbf{h}_i^l$ and $\mathbf{h}_j^l$, it only depends on squared distances $\|\mathbf{x}_i^l - \mathbf{x}_j^l\|^2$, which are E(n) invariant. Next, Equation 43 computes $\mathbf{x}_i^{l+1}$ by a weighted sum of differences $(\mathbf{x}_i - \mathbf{x}_j)$ which is added to $\mathbf{x}_i$, this transforms as a type-1 vector and preserves equivariance (see Appendix B.1). Finally the last two equations 44 and 45 that generate the next layer node-embeddings $\mathbf{h}_i^{l+1}$ remain E(n) invariant since they only depend on $\mathbf{h}_i^l$ and $\mathbf{m}_{ij}$ which, as we saw above, are E(n) invariant. Therefore the output $\mathbf{h}_i^{l+1}$ is E(n) invariant and $\mathbf{x}_i^{l+1}$ is E(n) equivariant to $\mathbf{x}_i^l$. Inductively, a composition of EGCLs will also be equivariant.

5.3.2 *Extending EGNNs for Vector Type Representations*

In this section we propose a slight modification to the presented method such that we explicitly keep track of the particle's momentum. In some scenarios this can be useful not only to obtain an estimate of the particle's velocity at every layer but also to provide an initial velocity value in those cases where it is not 0. We can include momentum $\mathbf{V}^l = \{\mathbf{v}_0^l, \ldots, \mathbf{v}_{M-1}^l\}$ to our proposed method by just replacing Equation 43 of our model with the following equation:

$$\begin{aligned} \mathbf{v}_i^{l+1} &= \phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^{\text{init}} + C\sum_{j \neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \\ \mathbf{x}_i^{l+1} &= \mathbf{x}_i^l + \mathbf{v}_i^{l+1} \end{aligned} \tag{46}$$

Note that this extends the EGCL layer as $\mathbf{H}^{l+1}, \mathbf{X}^{l+1}, \mathbf{V}^{l+1} = \text{EGCL}[\mathbf{H}^l, \mathbf{X}^l, \mathbf{V}^{\text{init}}, \mathcal{E}]$. The only difference is that now we broke down the coordinate update (Eq. 43) in two steps, first we compute the velocity $\mathbf{v}_i^{l+1}$ and then we use this velocity to update the position $\mathbf{x}_i^l$. The initial velocity $\mathbf{v}_i^{\text{init}}$ is scaled by a new function $\phi_v : \mathbb{R}^N \to \mathbb{R}^1$ that maps the node embedding $\mathbf{h}_i^l$ to a scalar value. Notice that if the initial velocity is set to zero ($\mathbf{v}_i^{\text{init}} = 0$), both equations 43 and 46 become

exactly the same. We proof the equivariance of this variant of the model in Appendix B.2.1. This variant is used in experiment 5.5.1 where we provide the initial velocity of the system, and predict a relative position change.

5.3.3 *Inferring the Edges*

Given a point cloud or a set of nodes, we may not always be provided with an adjacency matrix. In those cases we can assume a fully connected graph where all nodes exchange messages with each other $j \neq i$ as done in Equation 44. This fully connected approach may not scale well to large point clouds where we may want to locally limit the exchange of messages $\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}$ to a neighborhood $\mathcal{N}(i)$ to avoid an overflow of information.

Similarly to (Serviansky et al., 2020; Kipf et al., 2018), we present a simple solution to infer the relations/edges of the graph in our model, even when they are not explicitly provided. Given a set of neighbors $\mathcal{N}(i)$ for each node i, we can re-write the aggregation operation from our model (Eq. 44) in the following way:

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij} = \sum_{j \neq i} e_{ij} \mathbf{m}_{ij} \tag{47}$$

Where e_{ij} takes value 1 if there is an edge between nodes (i, j) and 0 otherwise. Now we can choose to approximate the relations e_{ij} with the following function $e_{ij} \approx \phi_{inf}(\mathbf{m}_{ij})$, where $\phi_{inf} : \mathbb{R}^{nf} \to [0, 1]$[1] resembles a linear layer followed by a sigmoid function that takes as input the current edge embedding and outputs a soft estimation of its edge value. This modification does not change the $E(n)$ properties of the model since we are only operating on the messages $\mathbf{m}_{ij}$ which are already $E(n)$ invariant.

5.4 RELATED WORK

Group equivariant neural networks have demonstrated their effectiveness in a wide variety of tasks (Cohen and Welling, 2016; Cohen and Welling, 2017; Weiler and Cesa, 2019; Rezende et al., 2019; Romero and Cordonnier, 2021). Recently, various forms and methods to achieve E(3) or SE(3) equivariance have been proposed. Thomas et al. (2018b) and Fuchs et al. (2020) utilize the spherical harmonics to compute a basis for the transformations, which allows transformations between higher-order representations. A downside to this method is that the spherical harmonics need to be recomputed which can be expensive. Currently, an extension of this method to arbitrary dimensions is unknown. Finzi et al. (2020b) parametrize transformations by mapping kernels on the Lie Algebra. For this method the neural network outputs are in certain situations stochastic, which may be undesirable. Horie et al. (2020) proposes a set of isometric invariant and equivairant transformations for Graph Neural Networks. Köhler et al. (2019) and

Table 5: Comparison of different works from the literature written under the message passing notation. We created this table with the aim of providing a clear and simple way to compare these different methods. The names from left to right are: Graph Neural Networks (Gilmer et al., 2017); Radial Field from Equivariant Flows (Köhler et al., 2019); Tensor Field Networks (Thomas et al., 2018b); Schnet (Schütt et al., 2017b); and our Equivariant Graph Neural Network. The rows in the table below the model name represent the edge operation, aggregation and node update respectively. The difference between two points is written as $\mathbf{r}_{ij} = (\mathbf{x}_i - \mathbf{x}_j)$. The layer index $l+1$ is written as l'.

GNN	Radial Field	TFN	Schnet	EGNN
$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l)$	$\mathbf{m}_{ij} = \phi_{rf}(\lvert\mathbf{r}_{ij}^l\rvert)\mathbf{r}_{ij}^l$	$\mathbf{m}_{ij} = \sum_k \mathbf{W}^{\ell k}\mathbf{r}_{ji}^l \mathbf{h}_i^{lk}$	$\mathbf{m}_{ij} = \phi_c(\lvert\mathbf{r}_{ij}^l\rvert)\phi_s(\mathbf{h}_j^l)$	$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \lvert\mathbf{r}_{ij}^l\rvert^2)$ $\hat{\mathbf{m}}_{ij} = \mathbf{r}_{ij}^l \phi_x(\mathbf{m}_{ij})$
$\mathbf{m}_i = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$ $\hat{\mathbf{m}}_i = C\sum_{j\neq i} \hat{\mathbf{m}}_{ij}$
$\mathbf{h}_i^{l'} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$	$\mathbf{x}_i^{l'} = \mathbf{x}_i^l + \mathbf{m}_i$	$\mathbf{h}_i^{l',\ell} = w^\ell \mathbf{h}_i^{l\ell} + \mathbf{m}_i$	$\mathbf{h}_i^{l'} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$	$\mathbf{h}_i^{l'} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$ $\mathbf{x}_i^{l'} = \mathbf{x}_i^l + \hat{\mathbf{m}}_i$
Non-equivariant	$E(n)$-Equivariant	$SE(3)$-Equivariant	$E(n)$-Invariant	$E(n)$-Equivariant

Köhler et al. (2020a) propose an $E(n)$ equivariant network to model 3D point clouds, but the method is only defined for positional data on the nodes without any feature dimensions.

Another related line of research concerns message passing algorithms on molecular data. (Gilmer et al., 2017) presented a message passing setting (or Graph Neural Network) for quantum chemistry, this method is permutation equivariant but not translation or rotation equivariant. (Kondor et al., 2018) extends the equivariance of GNNs such that each neuron transforms in a specific way under permutations, but this extension only affects its permutation group and not translations or rotations in a geometric space. Further works (Schütt et al., 2017b; Schütt et al., 2017a) build $E(n)$ invariant message passing networks by inputting the relative distances between points. Klicpera et al. (2020d) and Klicpera et al. (2020a) in addition to relative distances it includes a modified message passing scheme analogous to Belief Propagation that considers angles and directional information equivariant to rotations. It also uses Bessel functions and spherical harmonics to construct and orthogonal basis. Anderson et al. (2019) and Miller et al. (2020) include $SO(3)$ equivariance in its intermediate layers for modelling the behavior and properties of molecular data. Our method is also framed as a message passing framework but in contrast to these methods it achieves $E(n)$ equivariance.

Relationship with existing methods

In Table 5, the EGNN equations are detailed together with some of its closest methods from the literature under the message passing notation from (Gilmer et al., 2017). This table aims to provide a simple way to compare these different algorithms. It is structured in three main rows that describe i) the edge ii) aggregation and iii) node update operations. The GNN algorithm is the same as the previously introduced in Section 5.2.2. Our EGNN algorithm is also equivalent to the description in Section 5.3 but notation has been modified to match the (edge, aggregation, node) format. In all equations $\mathbf{r}_{ij}^l = (\mathbf{x}_i^l - \mathbf{x}_j^l)$. Notice that except the EGNN, all algorithms have the same aggregation operation and the main differences arise from the edge operation. The algorithm that we call "Radial Field" is the $E(n)$ equivariant update from (Köhler et al., 2019). This method is $E(n)$ equivariant, however its main limitation is that it only operates on $\mathbf{x}_i$ and it does not propagate node features $\mathbf{h}_i$ among nodes. In the method ϕ_{rf} is modelled as an MLP. Tensor Field Networks (TFN) (Thomas et al., 2018b) instead propagate the node embeddings $\mathbf{h}$ but it uses spherical harmonics to compute its learnable weight kernel $\mathbf{W}^{\ell k} : \mathbb{R}^3 \to \mathbb{R}^{(2\ell+1)\times(2k+1)}$ which preserves $SE(3)$ equivariance but is expensive to compute an limited to the 3 dimensional space. The SE(3) Transformer (Fuchs et al., 2020) (not included in this table), can be interpreted as an extension of TFN with attention. Schnet (Schütt et al., 2017b) can be interpreted as an $E(n)$ invariant Graph Neural Network where ϕ_c receives as input relative distances and outputs a continuous filter convolution that multiplies the neighbor embeddings $\mathbf{h}_i$. Our EGNN differs from these other methods in terms that it performs two different updates in each of the table rows, one related to the embeddings $\mathbf{h}_i$ and another related to the coordinates $\mathbf{x}_i$, these two variables exchange information in the edge operation. In summary the EGNN can retain the flexibility of GNNs while remaining $E(n)$ equivariant as the Radial Field algorithm and without the need to compute expensive operations (i.e. spherical harmonics).

5.5 EXPERIMENTS

5.5.1 *Modelling a Dynamical System | N-body System*

In a dynamical system a function defines the time dependence of a point or set of points in a geometrical space. Modelling these complex dynamics is crucial in a variety of applications such as control systems (Chua et al., 2018), model based dynamics in reinforcement learning (Nagabandi et al., 2018), and physical systems simulations (Grzeszczuk et al., 1998; Watters et al., 2017). In this experiment we forecast the positions for a set of particles which are modelled by simple interaction rules, yet can exhibit complex dynamics.

Similarly to (Fuchs et al., 2020), we extended the Charged Particles N-body experiment from (Kipf et al., 2018) to a 3 dimensional space. The system consists of 5 particles that carry a positive or negative charge and have a position and a velocity associated in 3-dimensional space. The system is controlled by physic rules: particles are attracted or repelled depending on their charges. This is an equivariant task since rotations and translations on the input set of particles result in the same transformations throughout the entire trajectory.

DATASET: We sampled 3.000 trajectories for training, 2.000 for validation and 2.000 for testing. Each trajectory has a duration of 1.000 timesteps. For each trajectory we are provided with the initial particle positions $\mathbf{P}^0 = \{\mathbf{p}_1^0, \dots \mathbf{p}_5^0\} \in \mathbb{R}^{5\times 3}$, their initial velocities $\mathbf{V}^0 = \{\mathbf{v}_1^0, \dots \mathbf{v}_5^0\} \in \mathbb{R}^{5\times 3}$ and their respective charges $\mathbf{c} = \{c_1, \dots c_5\} \in \{-1, 1\}^5$. The task is to estimate the positions of the five particles after 1.000 timesteps. We optimize the averaged Mean Squared Error of the estimated position with the ground truth one.

IMPLEMENTATION DETAILS: In this experiment we used the extension of our model that includes velocity from Section 5.3.2. We input the positions $\mathbf{p}_i^0$ as the first layer coordinates $\mathbf{x}_i^0$ of our model and the velocity $\mathbf{v}_i^0$ as the initial velocity in Equation 46, the norms $\|\mathbf{v}_i^0\|$ are also provided as features to $\mathbf{h}_i^0$ through a linear mapping. The charges are input as edge attributes $\mathbf{a}_{ij} := c_i c_j$. The model outputs the last layer coordinates $\mathbf{X}^L$ as the estimated positions. We compare our method to its non equivariant Graph Neural Network (GNN) cousin, and the equivariant methods: Radial Field (Köhler et al., 2019), Tensor Field Networks and the SE(3) Transformer. All algorithms are composed of 4 layers and have been trained under the same conditions, batch size 100, 10.000 epochs, Adam optimizer, the learning rate was tuned independently for each model. We used 64 features for the hidden layers in the Radial Field, the GNN and our EGNN. As non-linearity we used the Swish activation function (Ramachandran et al., 2017). For TFN and the SE(3) Transformer we swept over different number of vector types and features and chose those that provided the best performance. Further implementation details are provided in Appendix B.3.1. A Linear model that simply considers the motion equation $\mathbf{p}_i^{(t)} = \mathbf{p}_i^0 + \mathbf{v}_i^0 t$ is also included as a baseline. We also provide the average forward pass time in seconds for each of the models for a batch of 100 samples in a GTX 1080 Ti GPU.

RESULTS: As shown in Table 6 our model significantly outperforms the other equivariant and non-equivariant alternatives while still being efficient in terms of running time. It reduces the error with respect to the second best performing method by a 32%. In addition it doesn't require the computation of spherical harmonics which makes it more time efficient than Tensor Field Networks and the SE(3) Transformer.

Table 6: Mean Squared Error for the future position estimation in the N-body system experiment, and forward time in seconds for a batch size of 100 samples running in a GTX 1080Ti GPU.

Method	MSE	Forward time (s)
Linear	0.0819	.0001
SE(3) Transformer	0.0244	.1346
Tensor Field Network	0.0155	.0343
Graph Neural Network	0.0107	.0032
Radial Field	0.0104	.0039
EGNN	**0.0071**	.0062

ANALYSIS FOR DIFFERENT NUMBER OF TRAINING SAMPLES: We want to analyze the performance of our EGNN in the small and large data regime. In the following, we report on a similar experiment as above, but instead of using 3.000 training samples we generated a new training partition of 50.000 samples and we sweep over different amounts of data from 100 to 50.000 samples. We compare the performances of our EGNN vs its non-equivariant GNN counterpart and the Radial Field algorithm. Results are presented in Figure 12. Our method outperforms both Radial Field and GNNs in the small and large data regimes. This shows the EGNN is more data efficient than GNNs since it doesn't require to generalize over rotations and translations of the data while it ensembles the flexibility of GNNs in the larger data regime. Due to its high model bias, the Radial Field algorithm performs well when data is scarce but it is unable to learn the subtleties of the dataset as we increase the training size. In summary, our EGNN benefits from both the high bias of $E(n)$ methods and the flexibility of GNNs.

5.5.2 *Graph Autoencoder*

A Graph Autoencoder can learn unsupervised representations of graphs in a continuous latent space (Kipf and Welling, 2016b; Simonovsky and Komodakis, 2018). In this experiment section we use our EGNN to build an Equivariant Graph Autoencoder. We will explain how Graph Autoencoders can benefit from equivariance and we will show how our method outperforms standard GNN autoencoders in the provided datasets. This problem is particularly interesting since the embedding space can be scaled to larger dimensions and is not limited to a 3-dimensional Euclidean space.

Similarly to the work of (Kipf and Welling, 2016b) further extended by Section 3.3 in (Liu et al., 2019), our graph auto-encoder $\mathbf{Z} = q(\mathcal{G})$ embeds a graph $\mathcal{G}$

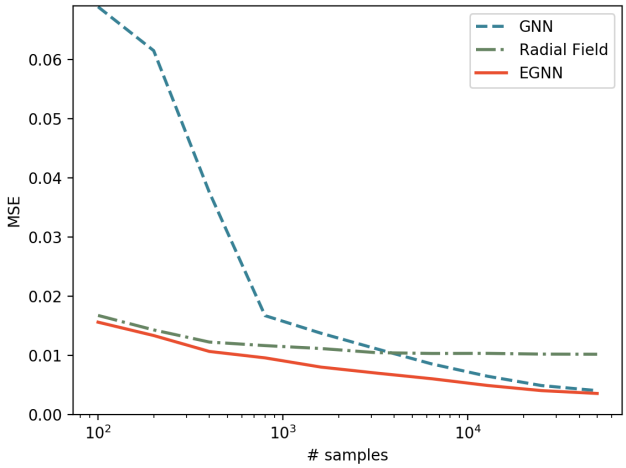

Figure 12: Mean Squared Error in the N-body experiment for the Radial Field, GNN and EGNN methods when sweeping over different amounts of training data.

into a set of latent nodes $\mathbf{Z} = \{\mathbf{z}_1, \dots \mathbf{z}_M\} \in \mathbb{R}^{M \times n}$, where M is the number of nodes and n the embedding size per node. Notice this may reduce the memory complexity to store the graphs from $O(M^2)$ to $O(Mn)$ where n may depend on M for a certain approximation error tolerance. This differs from the variational autoencoder proposed in (Simonovsky and Komodakis, 2018) which embeds the graph in a single vector $\mathbf{z} \in \mathbb{R}^K$, which causes the reconstruction to be computationally very expensive since the nodes of the decoded graph have to be matched again to the ground truth. In addition to the introduced graph generation and representation learning methods, it is worth mentioning that in the context of graph compression other methods (Candès and Recht, 2009) can be used.

More specifically, we will compare our Equivariant Graph Auto-Encoder in the task presented in (Liu et al., 2019) where a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node features $\mathbf{H} \in \mathbb{R}^{M \times nf}$ and adjacency matrix $\mathbf{A} \in \{0, 1\}^{M \times M}$ is embedded into a latent space $\mathbf{Z} = q(\mathbf{H}, \mathbf{A}) \in \mathbb{R}^{M \times n}$. Following (Kipf and Welling, 2016b; Liu et al., 2019), we are only interested in reconstructing the adjacency matrix $\mathbf{A}$ since the datasets we will work with do not contain node features. The decoder $g(\cdot)$ proposed by (Liu et al., 2019) takes as input the embedding space $\mathbf{Z}$ and outputs the reconstructed adjacency matrix $\hat{\mathbf{A}} = g(\mathbf{Z})$, this decoder function is defined as follows:

$$\hat{A}_{ij} = g_e(\mathbf{z}_i, \mathbf{z}_j) = \frac{1}{1 + \exp(w \left\|\mathbf{z}_i - \mathbf{z}_j\right\|^2 + b)} \tag{48}$$

Where w and b are its only learnable parameters and $g_e(\cdot)$ is the decoder edge function applied to every pair of node embeddings. It reflects that edge probabilities will depend on the relative distances among node embeddings. The training

loss is defined as the binary cross entropy between the estimated and the ground truth edges $\mathcal{L} = \sum_{ij} \text{BCE}(\hat{A}_{ij}, A_{ij})$.

THE SYMMETRY PROBLEM: The above stated autoencoder may seem straightforward to implement at first sight but in some cases there is a strong limitation regarding the symmetry of the graph. Graph Neural Networks are convolutions on the edges and nodes of a graph, i.e. the same function is applied to all edges and to all nodes. In some graphs (e.g. those defined only by its adjacency matrix) we may not have input features in the nodes, and for that reason the difference among nodes relies only on their edges or neighborhood topology. Therefore, if the neighborhood of two nodes is exactly the same, their encoded embeddings will be the same too. A clear example of this is a cycle graph (an example of a 4 nodes cycle graph is provided in Figure 13). When running a Graph Neural Network encoder on a node featureless cycle graph, we will obtain the exact same embedding for each of the nodes, which makes it impossible to reconstruct the edges of the original graph from the node embeddings. The cycle graph is a severe example where all nodes have the exact same neighborhood topology but these symmetries can be present in different ways for other graphs with different edge distributions or even when including node features if these are not unique.

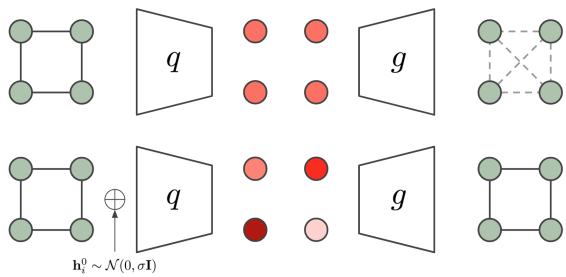

Figure 13: Visual representation of a Graph Autoencoder for a 4 nodes cycle graph. The bottom row illustrates that adding noise at the input graph breaks the symmetry of the embedding allowing the reconstruction of the adjacency matrix.

An ad-hoc method to break the symmetry of the graph is introduced by (Liu et al., 2019). This method introduces noise sampled from a Gaussian distribution into the input node features of the graph $\mathbf{h}_i^0 \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{I})$. This noise allows different representations for all node embeddings and as a result the graph can be decoded again, but it comes with a drawback, the network has to generalize over the new introduced noise distribution. Our Equivariant Graph Autoencoder will remain translation and rotation equivariant to this sampled noise which we find makes the generalization much easier. Another way of looking at this is considering the sampled noise makes the node representations go from structural to positional (Srinivasan and Ribeiro, 2019) where $\text{E}(n)$ equivariance may be beneficial. In our case we will simply input this noise as the input coordinates $\mathbf{X}^0 \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{I}) \in \mathbb{R}^{M \times n}$ of our EGNN which will output an equivariant transformation of them

Table 7: We report the Binary Cross Entropy, % Error and F1 scores for the test partition on the Graph Autoencoding experiment in the Community Small and Erdos&Renyi datasets.

	Community Small			Erdos&Renyi		
Encoder	BCE	% Error	F1	BCE	% Error	F1
Baseline	-	31.79	.0000	-	25.13	0.000
GNN	6.75	1.29	0.980	14.15	4.62	0.907
Noise-GNN	3.32	0.44	0.993	4.56	1.25	0.975
Radial Field	9.22	1.19	0.981	6.78	1.63	0.968
EGNN	2.14	0.06	0.999	1.65	0.11	0.998

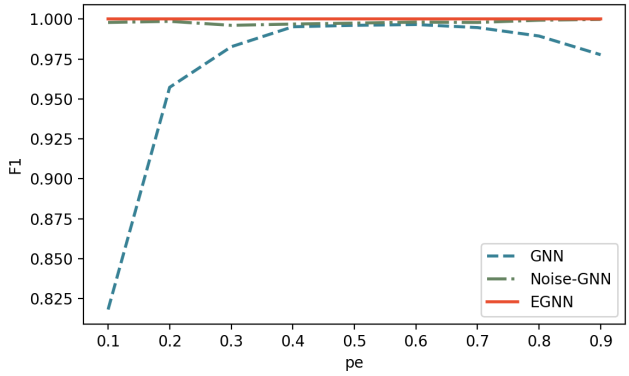

Figure 14: In this Figure we report the F1 score when overfitting a training partition of 100 samples in the Erdos&Renyi dataset for different values of sparsity p_e. The GNN is not able to successfully auto-encode sparse graphs (small p_e values) for the Erdos&Renyi dataset even when evaluating on the training set an overfitting task. This is due to the symmetry problem already discussed.

$\mathbf{X}^L$, this output will be used as the embedding of the graph (i.e. $\mathbf{Z} = \mathbf{X}^L$) which is the input to the decoder from Equation 48.

DATASET: We generated community-small graphs (You et al., 2018; Liu et al., 2019) by running the original code from (You et al., 2018). These graphs contain $12 \leqslant M \leqslant 20$ nodes. We also generated a second dataset using the Erdos&Renyi generative model (Bollobás, 1998) sampling random graphs with an initial number of $7 \leqslant M \leqslant 16$ nodes and edge probability $p_e = 0.25$. We sampled 5.000 graphs for training, 500 for validation and 500 for test for both datasets. Each graph is defined as and adjacency matrix $A \in \{0, 1\}^{M \times M}$.

IMPLEMENTATION DETAILS: Our Equivariant Graph Auto-Encoder is composed of an EGNN encoder followed by the decoder from Equation 48. The graph edges A_{ij} are input as edge attributes $\mathbf{a}_{ij}$ in Equation 42. The noise used

to break the symmetry is input as the coordinates $\sigma\ \mathbf{X}^0 \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{I}) \in \mathbb{R}^{M\times n}$ in the first layer and $\mathbf{H}^0$ is initialized as ones since we are working with featureless graphs. As mentioned before, the encoder outputs an equivariant transformation on the coordinates which is the graph embedding and input to the decoder $\mathbf{Z} = \mathbf{X}^L \in \mathbb{R}^{M\times n}$. We use $n = 8$ dimensions for the embedding space. We compare the EGNN to its GNN cousin, we also compare to Noise-GNN which is an adaptation of our GNN to match the setting from (Liu et al., 2019), and we also include the Radial Field algorithm as an additional baseline. All four models have 4 layers, 64 features for the hidden layers, the Swish activation function as a non-linearity and they were all trained for 100 epochs using the Adam optimizer and learning rate 10^{-4}. More details are provided in Appendix B.3.2. Since the number of nodes is larger than the number of layers, the receptive field of a GNN may not comprise the whole graph which can make the comparison unfair with our EGNN. To avoid this limitation, all models exchange messages among all nodes and the edge information is provided as edge attributes $\mathbf{a}_{ij} = A_{ij}$ in all of them.

RESULTS: In the table from Figure 7 we report the Binary Cross Entropy loss between the estimated and ground truth edges, the % Error which is defined as the percentage of wrong predicted edges with respect to the total amount of potential edges, and the F1 score of the edge classification, all numbers refer to the test partition. We also include a "Baseline" that predicts all edges as missing $\hat{A}_{ij} = 0$. The standard GNN seems to suffer from the symmetry problem and provides the worst performance. When introducing noise (Noise-GNN), both the loss and the error decrease showing that it is actually useful to add noise to the input nodes. Finally, our EGNN remains $E(n)$ equivariant to this noise distribution and provides the best reconstruction with a 0.11% error in the Erdos&Renyi dataset and close to optimal 0.06% in the Community Small dataset. A further analysis of the reconstruction error for different n embedding sizes is reported in Appendix B.4.1.

OVERFITTING THE TRAINING SET: We explained the symmetry problem and we showed the EGNN outperforms other methods in the given datasets. Although we observed that adding noise to the GNN improves the results, it is difficult to exactly measure the impact of the symmetry limitation in these results independent from other factors such as generalization from the training to the test set. In this section we conduct an experiment where we train the different models in a subset of 100 Erdos&Renyi graphs and embedding size $n = 16$ with the aim to overfit the data. We evaluate the methods on the training data. In this experiment the GNN is unable to fit the training data properly while the EGNN can achieve perfect reconstruction and Noise-GNN close to perfect. We sweep over different p_e sparsity values from 0.1 to 0.9 since the symmetry limitation is more present in very sparse or very dense graphs. We report the F1 scores of this experiment in the right plot of Figure 7.

In this experiment we showed that $E(n)$ equivariance improves performance when embedding graphs in a continuous space as a set of nodes in dimension n. Even though this is a simple reconstruction task, we think this can be a useful step towards generating graphs or molecules where often graphs (i.e. edges) are decoded as pairwise distances or similarities between nodes e.g. (Kipf and Welling, 2016b; Liu et al., 2019; Grover et al., 2019), and these metrics (e.g. Eq. 48) are $E(n)$ invariant. Additionally this experiment also showed that our method can successfully perform in a $E(n)$ equivariant task for higher dimensional spaces where $n > 3$.

5.5.3 *Molecular Data | QM9*

The QM9 dataset (Ramakrishnan et al., 2014) has become a standard in machine learning as a chemical property prediction task. The QM9 dataset consists of small molecules represented as a set of atoms (up to 29 atoms per molecule), each atom having a 3D position associated and a five dimensional one-hot node embedding that describe the atom type (H, C, N, O, F). The dataset labels are a variety of chemical properties for each of the molecules which are estimated through regression. These properties are invariant to translations, rotations and reflections on the atom positions. Therefore those models that are E(3) invariant are highly suitable for this task.

We imported the dataset partitions from (Anderson et al., 2019), 100K molecules for training, 18K for validation and 13K for testing. A variety of 12 chemical properties were estimated per molecule. We optimized and report the Mean Absolute Error between predictions and ground truth.

IMPLEMENTATION DETAILS: Our EGNN receives as input the 3D coordinate locations of each atom which are provided as $\mathbf{x}_i^0$ in Equation 42 and an embedding of the atom properties which is provided as input node features $\mathbf{h}_i^0$. Since this is an invariant task and also $\mathbf{x}_i^0$ positions are static, there is no need to update the particle's position $\mathbf{x}_i$ by running Equation 43 as we did in previous experiments. Consequently, we tried both manners and we didn't notice any improvement by updating $_i x$. When not updating the particle's position (i.e. skipping Equation 43), our model becomes $E(n)$ invariant, which is analogous to a standard GNN where all relative squared norms between pairs of points $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ are inputted to the edge operation (eq. 42). Additionally, since we are not provided with an adjacency matrix and molecules can scale up to 29 nodes, we use the extension of our model from Section 5.3.3 that infers a soft estimation of the edges. Our EGNN network consists of 7 layers, 128 features per hidden layer and the Swish activation function as a non-linearity. A sum-pooling operation preceded and followed by two layers MLPs maps all the node embeddings $\mathbf{h}_i^L$ from the output of the EGNN to the estimated property value. Further implementation details are reported in Appendix. B.3. We compare to NMP (Gilmer et al., 2017),

Table 8: Mean Absolute Error for the molecular property prediction benchmark in QM9 dataset. *DimeNet++ uses slightly different train/val/test partitions than the other works listed here.

Task	α	$\Delta\varepsilon$	ε_{HOMO}	ε_{LUMO}	μ	C_ν	G	H	R^2	U	U_0	ZPVE
Units	$bohr^3$	meV	meV	meV	D	cal/mol K	meV	meV	$bohr^3$	meV	meV	meV
NMP	.092	69	43	38	.030	.040	19	17	.180	20	20	1.50
Schnet	.235	63	41	34	.033	.033	14	14	.073	19	14	1.70
Cormorant	.085	61	34	38	.038	.026	20	21	.961	21	22	2.03
L1Net	.088	68	46	35	.043	.031	14	14	.354	14	13	1.56
LieConv	.084	49	30	25	.032	.038	22	24	.800	19	19	2.28
DimeNet++*	.044	33	25	20	.030	.023	8	7	.331	6	6	1.21
TFN	.223	58	40	38	.064	.101	-	-	-	-	-	-
SE(3)-Tr.	.142	53	35	33	.051	.054	-	-	-	-	-	-
EGNN	.071	48	29	25	.029	.031	12	12	.106	12	11	1.55

Schnet (Schütt et al., 2017b), Cormorant (Anderson et al., 2019), L1Net (Miller et al., 2020), LieConv (Finzi et al., 2020b), DimeNet++ (Klicpera et al., 2020a), TFN (Thomas et al., 2018b) and SE(3)-Tr. (Fuchs et al., 2020).

RESULTS: Results are presented in Table 8. Our method reports very competitive results in all property prediction tasks while remaining relatively simple, i.e. not introducing higher order representations, angles or spherical harmonics. Perhaps, surprisingly, we outperform other equivariant networks that consider higher order representations while in this task, we are only using type-0 representations (i.e. relative distances) to define the geometry of the molecules. In Appendix B.5 we prove that when only positional information is given (i.e. no velocity or higher order type features), then the geometry is completely defined by the norms in-between points up to $E(n)$-transformations, in other words, if two collections of points separated by $E(n)$ transformations are considered to be identical, then the relative norms between points is a unique identifier of the collection.

5.6 CONCLUSIONS

Equivariant graph neural networks are receiving increasing interest from the natural and medical sciences as they represent a new tool for analyzing molecules and their properties. In this chapter, we have introduced a new $E(n)$ equivariant deep architecture for graphs that is computationally efficient, easy to implement, and significantly improves over the current state-of-the-art on a wide range of tasks. We believe these properties make it ideally suited to make a direct impact on topics such as drug discovery, protein folding and the design of new materials, as well as applications in 3D computer vision.

In particular, after its publication (Satorras et al., 2021b), the Equivariant Graph Neural Network (EGNN) introduced in this chapter has been used in a wide variety of applications in the literature of molecular sciences as mentioned in the Subsequent Impact Section 1.3. And it will be our architecture of choice to build equivariant generative models in the following chapters. A benefit of the EGNN in the context of generative models is its high efficiency, which is particularly advantageous in the generative models used in the next chapters such as Normalizing Flows or and Diffusion Models that require multiple calls of the network to generate a single sample or a batch of samples.

Following the development of the EGNN, many new E(3) and SE(3) equivariant models have been proposed in literature providing better accuracy (e.g. Schütt et al., 2021; Thölke and De Fabritiis, 2022; Brandstetter et al., 2021). Nevertheless, the EGNN continues to be the architecture of choice in certain works due to its speed and simplicity as noted in Abanades et al., 2022.

Part III

EQUIVARIANT GENERATIVE MODELS

6

E(N) EQUIVARIANT NORMALIZING FLOWS

This Chapter is based on the content of:
Victor Garcia Satorras*, Emiel Hoogeboom*, Fabian B. Fuchs, Ingmar Posner, Max Welling (2021). "E(n) Equivariant Normalizing Flows." In: *Advances in Neural Information Processing Systems* 34

6.1 INTRODUCTION

In the previous chapter, we introduced an equivariant graph neural network for discriminative tasks, named the E(n) Equivariant Graph Neural Network (EGNN). In this and subsequent chapters, we build upon the EGNN to design equivariant generative models.

Generative models have made a significant impact in the domains of image and text generation (Ramesh et al., 2021), and they are promising candidates to impact the field of drug discovery through molecular modelling. Molecules inherently exist within a 3-dimensional space, subjecting them to Euclidean symmetries, such as rotations and translations. Leveraging these symmetries can enhance the generalization capabilities of generative models for molecular structures. For instance, forces on a molecule derived from its interatomic potentials are rotation equivariant with respect to the positional coordinates of their atoms.

As discussed in Chapter 5, considering these symmetries has substantially improved performance for discriminatory machine learning tasks on 3D coordinate data (Thomas et al., 2018b; Anderson et al., 2019; Finzi et al., 2020a; Fuchs et al., 2020; Klicpera et al., 2020b; Satorras et al., 2021b). However, for generative tasks, such as sampling new molecular structures, at the time of this work (Satorras et al., 2021a), the development of efficient yet powerful rotation equivariant approaches, despite making great progress, was still in its infancy.

In this chapter, we introduce a generative model equivariant to Euclidean symmetries: E(n) Equivariant Normalizing Flows (E-NFs). To construct E-NFs, we take the discriminative E(n) Graph Neural Networks and integrate them as a differ-

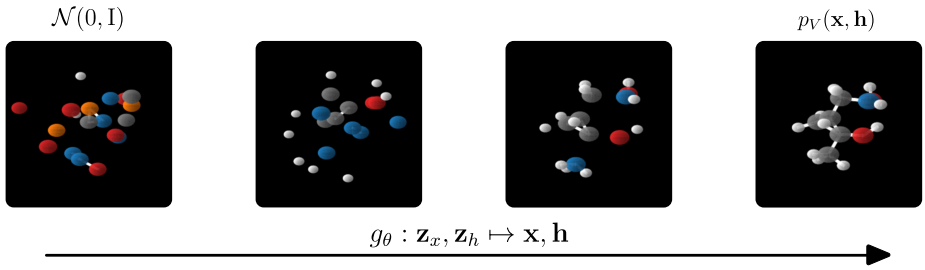

Figure 15: Overview of our method in the sampling direction. An equivariant invertible function g_θ has learned to map samples from a Gaussian distribution to molecules in 3D, described by $\mathbf{x}, \mathbf{h}$.

ential equation to obtain an invertible equivariant function. More specifically, we parameterize a continuous-time flow, where the first-order derivative is modeled by an EGNN. We also present minor improvements in the EGNN architecture to improve stability when used as a derivative. We show that our proposed flow model significantly outperforms its non-equivariant counterparts and previous equivariant generative methods (Köhler et al., 2020a). Additionally, we apply our method to molecule generation and demonstrate that our method can generate realistic molecules when trained on the QM9 dataset.

6.2 ACKNOWLEDGEMENT OF CONTRIBUTIONS

The work (Satorras et al., 2021a), upon which this chapter is based, was equally contributed to by Victor Garcia Satorras and Emiel Hoogeboom. Both authors contributed to the majority of the ideas, conducted the experiments, and wrote the paper. As for the parts of the model in Section 6.5 that define the Model Dynamics and Modeling Discrete Properties, these sections were primarily contributed by Victor and Emiel, respectively.

6.3 BACKGROUND

6.3.1 *Notation Clarification*

In the Background Section 2.1, we indicated that vectors are represented by boldface lowercase letters $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and matrices by boldface uppercase letters $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$. However, in this and the subsequent chapters, we slightly deviate from this notation and represent both in lowercase. That is, a set of points $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_M) \in \mathbb{R}^{M \times n}$ is represented as $\mathbf{x}$ instead of $\mathbf{X}$. We have chosen this notation in these chapters because our primary focus will be on the point cloud $\mathbf{x} \in \mathbb{R}^{M \times n}$ when defining the generative models, rather than the individual points/nodes $\mathbf{x}_i \in \mathbb{R}^n$. This choice is more consistent with the notation used in related literature on generative models (Kobyzev et al., 2020; Ho et al., 2020; Kingma et al., 2021).

6.3.2 *Normalizing Flows*

In Background Section 2.6 we discussed in high level the basics of a generative models and we explained how we can approximate a probability distribution by minimizing the negative log-likelihood of the data with respect to the model parameters. In this section we explain the basics of Normalizing Flows and how the optimization of the log-likelihood is formalized for this family of model.

A basic base distribution $p_Z(\mathbf{z})$, such as a normal distribution, together with a learnable invertible transformation $\mathbf{x} = g_\theta(\mathbf{z})$, results in a complex distribution $p_X(\mathbf{x})$. By defining f_θ as the inverse of g_θ, i.e., $f_\theta = g_\theta^{-1}$, we can calculate the likelihood of a data point $\mathbf{x}$ under p_X with the change of variables formula, as follows:

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left|\det J_f(\mathbf{x})\right|, \text{ where } \mathbf{z} = f(\mathbf{x}), \tag{49}$$

where J_f is the Jacobian of f_θ. A particular type of normalizing flows are *continuous-time* normalizing flows (Chen et al., 2017; Chen et al., 2018b; Grathwohl et al., 2018). These flows use a conceptual time direction to denote an invertible transformation in the form of a differential equation. The first order differential is predicted by a neural network ϕ, referred to as the dynamics. The formulation for continuous-time change of variables is given by:

$$\log p_X(\mathbf{x}) = \log p_Z(\mathbf{z}) + \int_0^1 \operatorname{Tr} J_\phi(\mathbf{x}(t))dt, \text{ where } \mathbf{z} = \mathbf{x} + \int_0^1 \phi(\mathbf{x}(t))dt, \tag{50}$$

Here, $\mathbf{x}(0) = \mathbf{x}$ and $\mathbf{x}(1) = \mathbf{z}$. In practice, the trace is estimated using Hutchinson's trace estimator as suggested by (Chen et al., 2018b; Grathwohl et al., 2018), and the integrals can be straightforwardly computed using the `torchdiffeq` package developed by Chen et al. (2018b). Regularizing the dynamics is often desired for faster training and more stable solutions (Finlay et al., 2020). In the context of designing a generative equivariant model, continuous-time normalizing flows are desirable because the constraints that need to be enforced on ϕ are relatively mild: ϕ only needs to be high order differentiable and Lipschitz continuous, with a possibly large Lipschitz constant.

6.3.3 *Equivariance in Normalizing Flows*

The concept of equivariance concerning Euclidean symmetries was introduced in the Background Section 2.5. In the context of probability distributions, the property of interest is often invariance, whereas for transformations g_θ we are interested in equivariance. To give an example, the likelihood of a molecule with

coordinates $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_M)$ should remain constant under any orientation, implying that $p(\mathbf{x}) = p(\mathbf{R}\mathbf{x})$, with no orientation being more likely than another.[1]

Specifically, if a function g_θ is equivariant and a base distribution $p_Z(\mathbf{z})$ is invariant, then the distribution p_X given by $\mathbf{x} = g_\theta(\mathbf{z})$ where $\mathbf{z} \sim p_Z(\mathbf{z})$ will also be invariant (Köhler et al., 2020a). Consequently, one can create expressive invariant distributions by combining an invariant base distribution p_Z with an equivariant function g_θ. Furthermore, when g_θ is restricted to be bijective and $f_\theta = g_\theta^{-1}$, then equivariance of g_θ implies equivariance of f_θ. Additionally, the likelihood p_X can be directly computed using the change of variables formula.

6.4 RELATED WORK

Group equivariant neural networks (Cohen and Welling, 2016; Cohen and Welling, 2017; Dieleman et al., 2016) have demonstrated their effectiveness in a wide variety of tasks. A growing body of literature is finding neural networks that are equivariant to transformations in Euclidean space (Thomas et al., 2018b; Fuchs et al., 2020; Horie et al., 2020; Finzi et al., 2020a; Hutchinson et al., 2020; Satorras et al., 2021b). These methods have proven their efficacy in discriminative tasks and modeling dynamical systems. On the other hand, graph neural networks (Bruna et al., 2013; Kipf and Welling, 2016a) can be seen as networks equivariant to permutations. Often, the methods that study Euclidean equivariance operate on point clouds embedded in Euclidean space, and so they also incorporate permutation equivariance.

Normalizing Flows (Rippel and Adams, 2013; Rezende and Mohamed, 2015b; Dinh et al., 2015) are an attractive class of generative models since they admit exact likelihood computation and can be designed for fast inference and sampling. Notably, Chen et al., 2018a; Chen et al., 2018b; Grathwohl et al., 2018 introduced continuous-time normalizing flows, a flow that is parametrized via a first-order derivative over time. This class of flows is useful because of the mild constraints on the parametrization function compared to other flow approaches (Dinh et al., 2017; Kingma and Dhariwal, 2018).

There are several specialized methods for molecule generation: Gebauer et al., 2019 generate 3D molecules iteratively via an autoregressive approach, but discretize positions and use additional focus tokens which makes them incomparable in log-likelihood. Gómez-Bombarelli et al., 2016; You et al., 2018; Liao et al., 2019 generate discrete graph structures instead of a coordinate space and Xu et al., 2021a generate molecule positions only. Some flows in the literature Noé

1 To be precise, in matrix multiplication notation $(\mathbf{R}\mathbf{x}_1, \ldots, \mathbf{R}\mathbf{x}_M) = \mathbf{x}\mathbf{R}^T$, for simplicity we use the notation $\mathbf{R}\mathbf{x} = (\mathbf{R}\mathbf{x}_1, \ldots, \mathbf{R}\mathbf{x}_M)$ and see $\mathbf{R}$ as an operation instead of a direct multiplication on the entire $\mathbf{x}$.

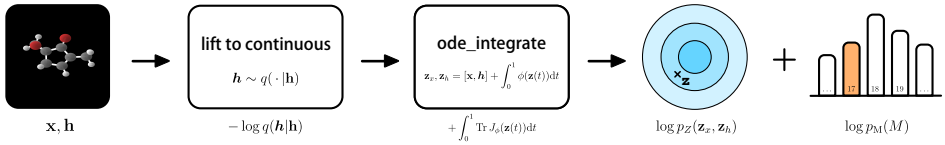

Figure 16: Overview of the training procedure: The discrete $\mathbf{h}$ is lifted to continuous $\mathbf{h}$. Then the variables $\mathbf{x}, \mathbf{h}$ are transformed by an ODE to $\mathbf{z}_x, \mathbf{x}_h$. To get a lower bound on $\log p_V(\mathbf{x}, \mathbf{h})$ we sum the variational term $-\log q(\mathbf{h}|\mathbf{h})$, the volume term from the ODE $\int_0^1 \mathrm{Tr}\, J_\phi(\mathbf{z}(t))dt$, the log-likelihood of the latent representation on a Gaussian $\log p_Z(\mathbf{z}_x, \mathbf{x}_h)$, and the log-likelihood of the molecule size $\log p_M(M)$. To train the model, the sum of these terms is maximized.

et al., 2018; Li et al., 2019; Köhler et al., 2020a model positional data, but not in combination with discrete properties.

Recently various forms of normalizing flows for equivariance have been proposed: Köhler et al., 2019; Köhler et al., 2020a propose flows for positional data with Euclidean symmetries, Rezende et al., 2019 introduce a Hamiltonian flow which can be designed to be equivariant using an invariant Hamiltonian function. Liu et al., 2019; Biloš and Günnemann, 2020 model distributions over graphs, with flows equivariant to permutations. Boyda et al., 2021 introduce equivariant flows for SU(n) symmetries. For adversarial networks, Dey et al., 2021 introduced a generative network equivariant to 90 degree image rotations and reflections. Our work differs from these approaches in that we design a general-purpose normalizing flow for sets of nodes that contain both positional *and* invariant features while remaining equivariant to E(n) transformations. This combination of positional and non-positional information results in a more expressive message passing scheme that defines the dynamics of our ODE. A by-product of our general approach is that it also allows us to jointly model features and structure of molecules in 3D space without any additional domain knowledge.

6.5 METHOD: E(N) EQUIVARIANT NORMALIZING FLOWS

In this section we propose our E(n) Equivariant Normalizing Flows, a probabilistic model for data with Euclidean symmetry. The generative process is defined by a simple invariant distribution $p_Z(\mathbf{z}_x, \mathbf{z}_h)$ (such as a Gaussian) over a latent representation of positions $\mathbf{z}_x \in \mathbb{R}^{M \times n}$ and a latent representation of invariant features $\mathbf{z}_h \in \mathbb{R}^{M \times nf}$ from which an invertible transformation $g_\theta(\mathbf{z}_x, \mathbf{z}_h) = (\mathbf{x}, \mathbf{h})$ generates $\mathcal{V} = (\mathbf{x}, \mathbf{h})$. These nodes consist of node features $\mathbf{h} \in \mathbb{R}^{M \times nf}$ and position coordinates $\mathbf{x} \in \mathbb{R}^{M \times n}$ embedded in a n-dimensional space. Together, p_Z and g_θ define a distribution p_V over node positions and invariant features.

Since we will restrict g_θ to be invertible, $f_\theta = g_\theta^{-1}$ exists and allows us to map from data space to latent space. This makes it possible to directly optimize the implied likelihood p_V of a datapoint $\mathcal{V}$ utilizing the change of variables formula:

$$p_V(\mathcal{V}) = p_V(\mathbf{x}, \mathbf{h}) = p_Z(f_\theta(\mathbf{x}, \mathbf{h}))|\det J_f| = p_Z(\mathbf{z}_x, \mathbf{z}_h)|\det J_f|, \tag{51}$$

where $J_f = \frac{d(\mathbf{z}_x, \mathbf{z}_h)}{d\mathcal{V}}$ is the Jacobian, where all tensors are vectorized for the Jacobian computation. The goal is to design the model p_V such that translations, rotations and reflections of $\mathbf{x}$ do not change $p_V(\mathbf{x}, \mathbf{h})$, meaning that p_V is E(n) invariant with respect to $\mathbf{x}$. Additionally, we want p_V to also be invariant to permutations of $(\mathbf{x}, \mathbf{h})$. A rich family of distributions can be learned by choosing p_Z to be invariant, and f_θ to be equivariant.

6.5.1 *The Normalizing Flow*

As a consequence, multiple constraints need to be enforced on f_θ. From a probabilistic modelling perspective 1) we require f_θ to be invertible, and from a symmetry perspective 2) we require f_θ to be equivariant. One could utilize the EGNN (Satorras et al., 2021b) which is E(n) equivariant, and adapt the transformation so that it is also invertible. The difficulty is that when both these constraints are enforced naïvely, the space of learnable functions might be small. For this reason we opt for a method that only requires mild constraints to achieve invertibility: neural ordinary differential equations. These methods require functions to be Lipschitz (which most neural networks are in practice, since they operate on a restricted domain) and to be continuously differentiable (which most smooth activation functions are).

To this extent, we define f_θ to be the differential equation, integrated over a conceptual time variable using a differential predicted by the EGNN ϕ. We redefine $\mathbf{x}$, $\mathbf{h}$ as functions depending on time, where $\mathbf{x}(t = 0) = \mathbf{x}$ and $\mathbf{h}(t = 0) = \mathbf{h}$ in the data space. Then $\mathbf{x}(1) = \mathbf{z}_x$, $\mathbf{h}(1) = \mathbf{z}_h$ are the latent representations. This admits a formulation of the flow f as the solution to an ODE defined as:

$$\mathbf{z}_x, \mathbf{z}_h = f(\mathbf{x}, \mathbf{h}) = [\mathbf{x}(0), \mathbf{h}(0)] + \int_0^1 \phi(\mathbf{x}(t), \mathbf{h}(t))dt. \tag{52}$$

The solution to this equation can be straightforwardly obtained by using the `torchdiffeq` package, which also supports backpropagation. The Jacobian term under the ODE formulation is $\log|\det J_f| = \int_0^1 \mathrm{Tr}\, J_\phi(\mathbf{x}(t), \mathbf{h}(t))dt$ as explained in Section 6.3, Equation 50. The Trace of J_ϕ has been approximated with the Hutchinson's trace estimator.

6.5.2 *The Dynamics*

The dynamics function ϕ in Equation 52 models the first derivatives of $\mathbf{x}$ and $\mathbf{h}$ with respect to time, over which we integrate. Specifically: $\frac{d}{dt}\mathbf{x}(t), \frac{d}{dt}\mathbf{h}(t) = \phi(\mathbf{x}(t), \mathbf{h}(t))$. This derivative is modelled by the EGNN of L layers introduced in Chapter 5:

$$\frac{d}{dt}\mathbf{x}(t), \frac{d}{dt}\mathbf{h}(t) = \mathbf{x}^L(t) - \mathbf{x}(t), \mathbf{h}^L(t) \quad \text{where} \quad \mathbf{x}^L(t), \mathbf{h}^L(t) = \text{EGNN}[\mathbf{x}(t), \mathbf{h}(t)]. \tag{53}$$

Notice that we directly consider the output $\mathbf{h}^L$ from the last layer L of the EGNN as the differential $\frac{d}{dt}\mathbf{h}(t)$ of the node features, since the representation is invariant. In contrast, the differential of the node coordinates is computed as the difference between the EGNN output and intput $\frac{d}{dt}\mathbf{x}(t) = \mathbf{x}^L - \mathbf{x}(t)$. This choice is consistent with the nature of velocity-type equivariance: although $\frac{d}{dt}\mathbf{x}(t)$ rotates exactly like $\mathbf{x}$, it is unaffected by translations as desired when computing velocities or forces.

The original EGNN equations from (Satorras et al., 2021b) are unstable when utilized in an ODE because the coordinate update from Equation 43 would easily explode. To counter this, we propose a simple extension in Equation 54 that normalizes the relative difference of two coordinates $\mathbf{x}_i^l - \mathbf{x}_j^l$ by their norm plus a constant C to ensure differentiability. In practice we set $C = 1$ and found this to give stable results. The full definition of the EGNN, considering this update, is provided in Appendix C.1.

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \sum_{j \neq i} \frac{(\mathbf{x}_i^l - \mathbf{x}_j^l)}{\|\mathbf{x}_i^l - \mathbf{x}_j^l\| + C} \phi_x\left(\mathbf{m}_{ij}\right) \tag{54}$$

6.5.3 *Translation Invariance*

Recall that we want the distribution $p_V(\mathcal{V})$ to be translation invariant with respect to the overall location and orientation of positional coordinates $\mathbf{x}$. For simplicity, let's assume only a distribution $p_X(\mathbf{x})$ over positions and an invertible function $\mathbf{z} = f(\mathbf{x})$. Translation invariance is defined as $p_X(\mathbf{x}+\mathbf{t}) = p_X(\mathbf{x})$ for all $\mathbf{t}$: a constant function. However, this cannot be a distribution since it cannot integrate to one. Instead, we have to restrict p_X to a subspace.

To construct a translation invariant p_X, we can restrict the data, flow f_θ and prior p_Z to a translation invariant linear subspace, for instance by centering the nodes so that their center of gravity is zero. Then the positions $\mathbf{x} \in \mathbb{R}^{M \times n}$ lie on the $(M-1) \times n$-dimensional linear subspace defined by $\sum_{i=1}^{M} \mathbf{x}_i = \mathbf{0}$. However, from a modelling perspective it is easier to represent node positions as M sets of coordinates that are n-dimensional in the ambient space. In short, we desire

the distribution to be defined on the subspace, but with the representation of the nodes in the ambient space. To limit the flow to the subspace, in practice only the mean of the output of the dynamics network ϕ is removed, following (Köhler et al., 2020a). Additionally, we can find the proper normalization constant for the base distribution.

6.5.4 *The Base Distribution*

Next we need to define a base distribution p_Z. This base distribution can be divided in two parts: the positional part $p(\mathbf{z}_x)$ and the feature part $p(\mathbf{z}_h)$. We will choose these parts to be independent so that $p(\mathbf{z}_x, \mathbf{z}_h) = p(\mathbf{z}_x) \cdot p(\mathbf{z}_h)$. The feature part is straightforward because the features are already invariant with respect to $E(n)$ symmetries, and only need to be permutation invariant. A common choice is a standard Gaussian $p(\mathbf{z}_h) = \mathcal{N}(\mathbf{z}_h|0, \mathbf{I})$. For the positional part recall that $\mathbf{z}_x$ lies on an $(M-1)n$ subspace, and we need to specify the distribution over this space. Standard Gaussian distributions are reflection and rotation invariant since $\|\mathbf{R}\mathbf{z}_x\|^2 = \mathbf{z}_x^T\mathbf{R}^T\mathbf{R}\mathbf{z}_x = \mathbf{z}_x^T\mathbf{R}^T\mathbf{R}\mathbf{z}_x = \|\mathbf{z}_x\|^2$ for any rotation or reflection $\mathbf{R}$. Further more, our particular projection $\tilde{\mathbf{z}}_x = P\mathbf{z}_x$ lies in a subspace $\sum_{i=1}^{M} \mathbf{z}_{x_i} = 0$ in which the point cloud is centered at the origin of coordinates. A valid choice for a rotation invariant base distribution on the subspace is given by:

$$p(\tilde{\mathbf{z}}_x) = \mathcal{N}(\tilde{\mathbf{z}}_x|0, \mathbf{I}) = \frac{1}{(2\pi)^{(M-1)n/2}} \exp\Big(-\frac{1}{2}\|\mathbf{z}_x\|^2\Big), \tag{55}$$

Which can be directly computed in the ambient space using $\|\mathbf{z}_x\|^2$, with the important property that the normalization constant uses the dimensionality of the subspace: $(M-1)n$, so that the distribution is properly normalized. A more formal proof to derive this distribution can be found in (Satorras et al., 2021a).

6.5.5 *Modelling Discrete Properties*

Normalizing flows model continuous distributions. However, the node features $\mathbf{h}$ may contain both ordinal (e.g. charge) and categorical (e.g. atom type) features. To train a normalizing flow on these, the values need to be lifted to a continuous space. Let $\mathbf{h} = (\mathbf{h}_{ord}, \mathbf{h}_{cat})$ be divided in ordinal and categorical features. For this we utilize variational dequantization (Ho et al., 2019) for the ordinal features and argmax flows (Hoogeboom et al., 2021) for the categorical features. For the ordinal representation $\mathbf{h}_{ord}$, interval noise $\mathbf{u} \sim q_{ord}(\cdot|\mathbf{h}_{ord})$ is used to lift to the continuous representation $\boldsymbol{h}_{ord} = \mathbf{h}_{ord} + \mathbf{u}$. Similarly, $\mathbf{h}_{cat}$ is lifted using a distribution $\boldsymbol{h}_{cat} \sim q_{cat}(\cdot|\mathbf{h}_{cat})$ where q_{cat} is the probabilistic inverse to an argmax function. Both q_{ord} and q_{cat} are parametrized using normal distributions where the mean and standard deviation are learned using an EGNN conditioned on the discrete representation. This formulation allows training on the continuous

representation $\boldsymbol{h} = (\boldsymbol{h}_{\text{ord}}, \boldsymbol{h}_{\text{cat}})$ as it lowerbounds an implied log-likelihood of the discrete representation $\mathbf{h}$ using variational inference:

$$\log p_H(\mathbf{h}) \geqslant \mathbb{E}_{\boldsymbol{h} \sim q_{\text{ord,cat}}(\,\cdot\,|\mathbf{h})} \Big[\log p_H(\boldsymbol{h}) - \log q_{\text{ord,cat}}(\boldsymbol{h}|\mathbf{h}) \Big] \tag{56}$$

To sample the discrete $\mathbf{h} \sim p_H$, first sample the continuous $\boldsymbol{h} \sim p_H$ via a flow and then compute $\mathbf{h} = (\text{round}(\boldsymbol{h}_{\text{ord}}), \text{argmax}(\boldsymbol{h}_{\text{cat}}))$ to obtain the discrete version. In short, instead of training directly on the discrete properties $\mathbf{h}$, the properties are lifted to the continuous variable $\boldsymbol{h}$. The lifting method depends on whether a feature is categorical or ordinal. On this lifted continuous variable $\boldsymbol{h}$ the flow learns p_H, which is guaranteed to be a lowerbound via Equation 56 on the discrete p_H. To avoid clutter, in the remainder of this chapter no distinction is made between $\mathbf{h}$ and $\boldsymbol{h}$ as one can easily transition between them using $q_{\text{ord}}, q_{\text{cat}}$ and the round, argmax functions.

6.5.6 *Modelling the Number of Nodes*

Finally, the number of nodes M may differ depending on the data. In this case we extend the model using a simple one dimensional categorical distribution p_M of M categories. This distribution p_M is constructed by counting the number of molecules and dividing by the total. The likelihood of a set of nodes is then $p_V(\mathbf{x}, \mathbf{h}, M) = p_{V_M}(\mathbf{x}, \mathbf{h}|M)p_M(M)$, where $p_{V_M}(\mathbf{x}, \mathbf{h}|M)$ is modelled by the flow as before and the same dynamics can be shared for different sizes as the EGNN adapts to the number of nodes. In notation we sometimes omit the M conditioning for clarity. To generate a sample, we first sample $M \sim p_M$, then $\mathbf{z}_x, \mathbf{z}_h \sim p_Z(\mathbf{z}_x, \mathbf{z}_h|M)$ and finally transform to the node features and positions via the flow.

6.6 EXPERIMENTS

6.6.1 *DW4 and LJ13*

In this section we study two relatively simple systems, DW-4 and LJ-13, presented in (Köhler et al., 2020a) where E(n) symmetries are present. These datasets have been synthetically generated by sampling from their respective energy functions using Markov Chain Monte Carlo (MCMC).

DW4: This system consists of only M=4 particles embedded in a 2-dimensional space which are governed by an energy function that defines a coupling effect between pairs of particles with multiple metastable states. More details are provided in Appendix C.2.

LJ-13: This is the second dataset used in (Köhler et al., 2020a) which is given by the *Leonnard-Jones* potential. It is an approximation of inter-molecular pair potentials that models repulsive and attractive interactions. It captures essential

Table 9: Negative Log Likelihood comparison on the test partition over different methods on DW4 and LJ13 datasets for different amount of training samples averaged over 3 runs.

	DW4				**LJ13**			
# Samples	10^2	10^3	10^4	10^5	10	10^2	10^3	10^4
GNF	11.93	11.31	10.38	7.95	43.56	42.84	37.17	36.49
GNF-att	11.65	11.13	9.34	7.83	43.32	36.22	33.84	32.65
GNF-att-aug	8.81	8.31	7.90	7.61	41.09	31.50	30.74	30.93
Simple dynamics	9.58	9.51	9.53	9.47	33.67	33.10	32.79	32.99
Kernel dynamics	8.74	8.67	8.42	8.26	35.03	31.49	31.17	31.25
E-NF	**8.31**	**8.15**	**7.69**	**7.48**	**33.12**	**30.99**	**30.56**	**30.41**

physical principles and it has been widely studied to model solid, fluid and gas states. The dataset consists of M=13 particles embedded in a 3-dimensional state. More details are provided in Appendix C.2.

Both energy functions (DW4 and LJ13) are equivariant to translations, rotations and reflections which makes them ideal to analyze the benefits of equivariant methods when E(n) symmetries are present on the data. We use the same MCMC generated dataset from (Köhler et al., 2020a). For both datasets we use 1,000 validation samples, and 1,000 test samples. We sweep over different numbers of training samples $\{10^2, 10^3, 10^4, 10^5\}$ and $\{10, 10^2, 10^3, 10^4\}$ for DW4 and LJ13 respectively to analyze the performance in different data regimes.

IMPLEMENTATION DETAILS: We compare to the state-of-the art E(n) equivariant flows "Simple Dynamics" and "Kernel Dynamics" presented in (Köhler et al., 2020a). We also compare to non-equivariant variants of our method, Graph Normalizing Flow (GNF), GNF with attention (GNF-att) and GNF with attention and data augmentation (GNF-att-aug), i.e. augmenting the data with rotations. Our E-NF method and its non-equivariant variants (GNF, GNF-att, GNF-att-aug) consist of 3 layers each, 32 features per layer, and SiLU activation functions. All reported numbers have been averaged over 3 runs. Further implementation details are provided in the Appendix C.2.

RESULTS: In Table 9 we report the cross-validated Negative Log Likelihood for the test partition. Our E-NF outperforms its non-equivariant variants (GNF, GNF-att and GNF-att-aug) and (Köhler et al., 2020a) methods in all data regimes. It is interesting to see the significant increase in performance when including data augmentation (from GNF-att to GNF-att-aug) in the non-equivariant models.

6.6.2 *QM9 Positional*

We introduce QM9 Positional as a subset of QM9 that only considers positional information and does not encode node features. The aim of this experiment is to compare our method to those that only operate on positional data (Köhler et al., 2020a) while providing a more challenging scenario than synthetically generated datasets. QM9 Positional consists only of molecules with 19 atoms/nodes, where each node only has a 3-dimensional positional vector associated. The likelihood of a molecule should be invariant to translations and rotations on a 3-dimensional space which makes equivariant models very suitable for this type of data. The dataset consists of 13831 training samples, 2501 for validation and 1813 for test.

In this experiment, in addition to reporting the estimated Negative Log Likelihood, we designed a a metric to get an additional insight into the quality of the generated samples. More specifically, we produce a histogram of relative distances between all pairs of nodes within each molecule and we compute the Jensen–Shannon divergence (Lin, 1991) $\mathrm{JS}_{\mathrm{div}}(\mathrm{P}_{\mathrm{gen}} \| \mathrm{P}_{\mathrm{data}})$ between the normalized histograms from the generated samples and from the training set. See Figure 17 for an example.

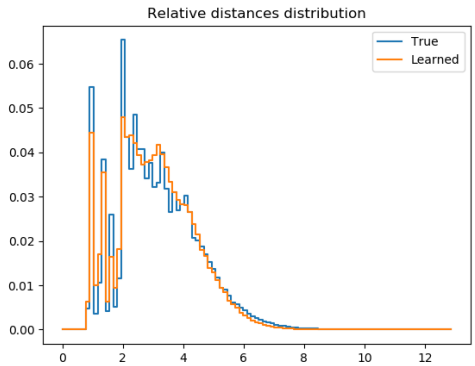

Figure 17: Normalized histogram of relative distances between atoms for QM9 Positional and E-NF generated samples.

IMPLEMENTATION DETAILS: As in the previous experiment, we compare our E-NF to its non-equivariant variants GNF, GNF-att, GNF-att-aug and to the equivariant methods from (Köhler et al., 2020a) Simple Dynamics and Kernel Dynamics. The dynamics of our E-NF, GNF, GNF-att and GNF-att-aug consist of 6 convolutional layers each, the number of features is set to 64 and all activation layers are SiLUs. The learning rate is set to $5 \cdot 10^{-4}$ for all experiments except for the E-NF and Simple dynamics which was set to $2 \cdot 10^{-4}$ for stability reasons. All experiments have been trained for 160 epochs. The JS divergence values have been averaged over the last 10 epochs for all models.

Table 10: Negative Log Likelihood (NLL) $-\log p_V(\mathbf{x})$ for the QM9 Positional dataset on the test data.

# Metrics	NLL	JS(rel. dist)
Simple dynamics	73.0	.086
Kernel dynamics	38.6	.029
GNF	-00.9	.011
GNF-att	-26.6	.007
GNF-att-aug	-33.5	.006
E-NF (ours)	**-70.2**	.006

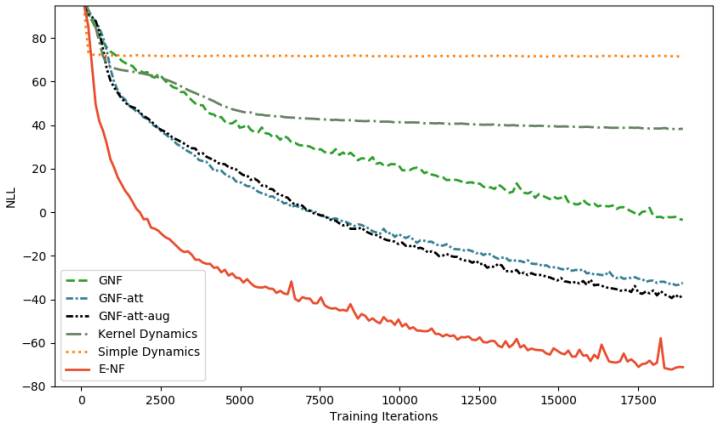

Figure 18: Training Negative Log Likelihood (NLL) curves for all methods.

RESULTS: In the table from Figure 18 we report the cross validated Negative Log Likelihood $-\log p_V(\mathbf{x})$ for the test data and the Jensen-Shannon divergence. Our E-NF outperforms all other algorithms in terms of NLL of the dataset. Additionally, the optimization curve with respect to the number of iterations converges much quicker for our E-NF than for the other methods as shown on the right in Figure 18. Regarding the JS divergence, the E-NF and GNF-att-aug achieve the best performance.

6.6.3 *QM9 Molecules*

QM9 (Ramakrishnan et al., 2014) is a molecular dataset standarized in machine learning as a chemical property prediction benchmark. It consists of small molecules (up to 29 atoms per molecule). Atoms contain positional coordinates embedded in a 3D space, a one-hot encoding vector that defines the type of molecule (H, C, N, O, F) and an integer value with the atom charge. Instead of predicting properties from molecules, we use the QM9 dataset to learn a generative distri-

bution over molecules. We desire the likelihood of a molecule to be invariant to translations and rotations, therefore, our E(n) equivariant normalizing flow is very suitable for this task. To summarize, we model a distribution over 3D positions $\mathbf{x}$, and atom properties $\mathbf{h}$. These atom properties consist of the atom type (categorical) and atom charge (ordinal).

We use the dataset partitions from (Anderson et al., 2019), 100K / 18K / 13K for training/validation/test respectively. To train the method, the nodes $(\mathbf{x}, \mathbf{h})$ are put into Equation 52 as $\mathbf{x}(0), \mathbf{h}(0)$ at time 0 and integrated to time 1. Using the continuous-time change of variables formula and the base distribution, the (negative) log-likelihood of a molecule is computed $-\log p_V(\mathbf{x}, \mathbf{h}, M)$. Since molecules differ in size, this term includes $-\log p_M(M)$ which models the number of atoms as a simple 1D categorical distribution and is part of the generative model as described in Section 6.5.

IMPLEMENTATION DETAILS: We compare our E-NF to the non-equivariant GNF-att and GNF-att-aug introduced in previous experiments. Notice in this experiment we do not compare to (Köhler et al., 2020a) since this dataset contains invariant feature data in addition to positional data. Each model is composed of 6 layers, 128 features in the hidden layers and SilU activation functions. We use the same learning rates as in QM9 Positional. Note that the baselines can be seen as instances of permutation equivariant flows (Liu et al., 2019; Biloš and Günnemann, 2020) but where the GNN architectures have been chosen to be as similar as possible to the architecture in our E-NFs.

Table 11: Neg. log-likelihood $-\log p_V(\mathbf{x}, \mathbf{h}, M)$, atom stability and mol stability for the QM9 dataset.

# Metrics	NLL	Atom stability	Mol stable
GNF-attention	-28.2	72%	0.3%
GNF-attention-augmentation	-29.3	75%	0.5%
E-NF (ours)	**-59.7**	**85%**	**4.9%**
Data	-	99%	95.2%

RESULTS (QUANTITATIVE): Results are reported in Table 11. As in previous experiments, our E-NF significantly outperforms the non-equivariant models GNF and GNF-aug. In terms of negative log-likelihood, the E-NF performs much better than its non-equivariant counterparts. One factor that increases this difference is the E-NFs ability to capture the very specific distributions of inter-atomic distances. Since the E-NF is able to better capture these sharp peaks in the distribution, the negative log-likelihood becomes much lower. This effect is also seen when studying the number of stable atoms and molecules, which is very sensitive to the inter-atomic distances. This stablity metric was computed over 10.000

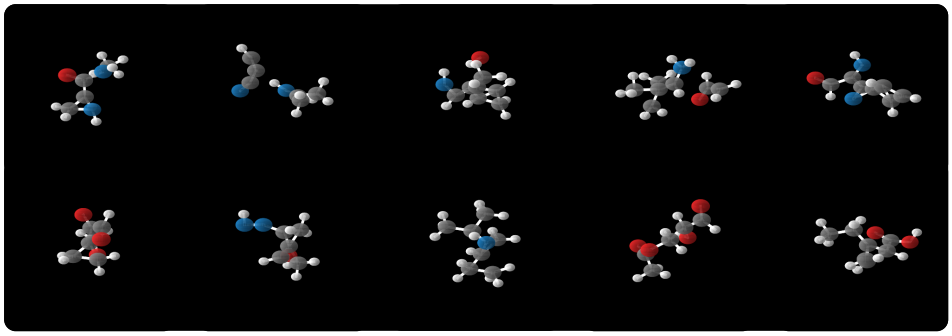

Figure 19: Sampled molecules by our E-NF. The top row contains random samples, the bottom row also contains samples but selected to be stable. Edges are drawn depending on inter-atomic distance.

samples from the model, for a detailed explanation of stability see Appendix C.4. Observe that it might also be possible to utilize post-processing to increase the molecule stability using prior knowledge. However, here we are mainly using this metric to see how many stable atoms and molecules the E-NF is able to generate in one shot, only by learning the molecule distribution. The E-NF on average produces 85% valid atoms, whereas the best baseline only produces 75% valid atoms. An even stronger improvement is visible when comparing molecule stability: where the E-NF produces 4.9% stable molecules versus 0.5% by the best baseline. Interestingly, the percentage of stable molecules is much lower than that of atoms. This is not unexpected: if even one atom in a large molecule is unstable, the entire molecule is considered to be unstable.

Finally, we evaluate the Validity, Uniqueness and Novelty as defined in (Simonovsky and Komodakis, 2018) for the generated molecules that are stable. For this purpose, we map the 3-dimensional representation of stable molecules to a graph structure and then to a SMILES notation using the rdkit toolkit. All our stable molecules are already defined as valid, therefore we only report the Novelty and Uniqueness since the Validity of those molecules that are already stable is 100%. The Novelty is defined as the ratio of stable generated molecules not present in the training set and Uniqueness is the ratio of unique stable generated molecules. Notice that different generated point clouds in 3-dimensional space may lead to the same molecule in the graph space or SMILES notation. Therefore, even if in the 3D space, all our generated samples were unique and novel, the underlying molecule that they represent doesn't have to be. Using our E-NF, we generated 10.000 examples to compute these metrics. We obtained 491 stable molecules (4.91%), from these subset 99.80% were unique and 93.28% were novel. Further analyses are provided in Appendix C.5.

RESULTS (QUALITATIVE): In Figure 19 samples from our model are shown. The top row contains random samples that have not been cherry-picked in any

way. Note that although the molecule structure is mostly accurate, sometimes small mistakes are visible: Some molecules are disconnected, and some atoms in the molecules do not have the required number of bonds. In the bottom row, random samples have been visualized but under the condition that the sample is stable. Note that atoms might be double-bonded or triple-bonded, which is indicated in the visualization software. For example, in the molecule located in the bottom row 4th column, an oxygen atom is double bonded with the carbon atom at the top of the molecule.

6.7 CONCLUSIONS, LIMITATIONS AND FUTURE WORK

In this chapter, we have introduced E(n) Equivariant Normalizing Flows (E-NFs): a generative model equivariant to Euclidean symmetries. E-NFs are continuous-time normalizing flows that use an EGNN with improved stability as parametrization. We have demonstrated that E-NFs considerably outperform existing normalizing flows in log-likelihood on DW4, LJ13, QM9, and also in the stability of generated molecules and atoms.

Despite the positive results, there are some limitations in the proposed model that could be addressed in future work: 1) The ODE type of flow makes training computationally expensive, as the same forward operation must be performed multiple times sequentially to solve the ODE equation. 2) The combination of the ODE with the EGNN exhibited some instabilities that we had to address (Equation 54). Although the model was stable in most experiments, we observed some rare peaks in the loss of the third experiment (QM9) that caused divergence in one very rare instance. 3) Our likelihood estimation is invariant to reflections, meaning our model assigns the same likelihood to both a molecule and its mirrored version. This could be advantageous or disadvantageous depending on whether we wish to assign distinct likelihoods to chiral molecules.

In the following chapter, we propose a new equivariant generative model based on denoising-diffusion models. This model addresses the issues (1) and (2) raised here and provides additional benefits such as greatly improved performance. Specifically, 1) it eliminates the need to forward the same sample multiple times sequentially during training, allowing for faster trainings. 2) The instabilities observed in the combination of the ODE with the EGNN are absent when employing the denoising-diffusion model.

7

EQUIVARIANT DIFFUSION FOR MOLECULE GENERATION IN 3D

This Chapter is based on the content of:
Emiel Hoogeboom*, Victor Garcia Satorras*, Clément Vignac*, Max Welling (2022). "Equivariant diffusion for molecule generation in 3D." In: *International conference on machine learning*. PMLR, pp. 8867–8887

7.1 INTRODUCTION

This chapter introduces an E(3) Equivariant Diffusion Model (EDM) for molecule generation in 3D. This model builds upon the EGNN from Chapter 5, handling both continuous coordinates and categorical data (atom types). And it addresses some of the main limitations present in the previous E(n) Equivariant Flows (E-NF) from Chapter 6. Particularly, it trains faster than E-NF, resulting in better scalability and significantly improved results compared to previous models in literature. To the best of our knowledge, this is the first diffusion model that directly generates molecules in 3D space.

Deep learning methods are influencing the field of molecular sciences as seen with the success of AlphaFold in protien folding prediction (AlQuraishi, 2019) along with a growing literature on silico molecule synthesis. (Simonovsky and Komodakis, 2018; Gebauer et al., 2019; Klicpera et al., 2020c; Simm et al., 2021). Molecules inherently exist within a 3-dimensional space, subjecting them to Euclidean symmetries, such as rotations and translations. Leveraging these symmetries can enhance the generalization capabilities of generative models for molecular structures and it has been extensively studied in (Thomas et al., 2018a; Fuchs et al., 2020; Finzi et al., 2020a)

Previous literature has explored E(3) equivariant layers in generative models such as autoregressive models (Gebauer et al., 2018; Gebauer et al., 2019) which introduce an artificial order in the atoms and are known to be difficult to scale during sampling (Xu et al., 2021c). Alternatively, continuous-time normalizing flows

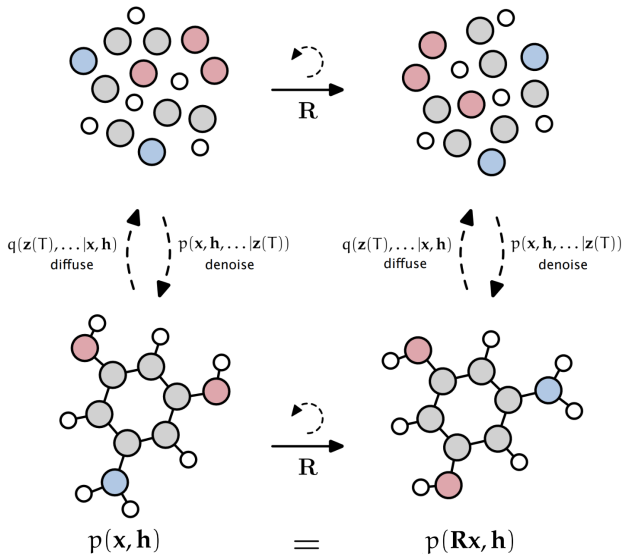

Figure 20: Overview of the EDM. To generate a molecule, a normal distributed set of points is denoised into a molecule consisting of atom coordinates $\mathbf{x}$ in 3D and atom types $\mathbf{h}$. As the model is rotation equivariant, the likelihood is preserved when a molecule is rotated by $\mathbf{R}$.

such as Köhler et al. (2020b) or E-NF (Satorras et al., 2021a) proposed in the previous chapter are expensive to train due to the need to integrate a differential equation, leading to limited performance and scalability.

Our proposed E(3) Equivariant Diffusion Model (EDM) learns to denoise a diffusion process operating on both continuous coordinates and categorical atom types. EDM does not require a particular atom ordering unlike autoregressive models, and can be trained much more efficiently than normalizing flows. To give an example, when trained on QM9, EDMs generate up to 16 times more stable molecules than E-NFs while requiring half of the training time. This efficiency in scaling allows EDMs to be trained on larger drug-like datasets such as GEOM-Drugs (Axelrod and Gomez-Bombarelli, 2020).

In summary, this chapter introduces a new equivariant denoising diffusion model that operates on both atom coordinates and categorical features. We apply this model to molecule generation, outperforming previous generative models in the literature. Additionally, we demonstrate that we can generate molecules with certain desired properties when conditioned accordingly.

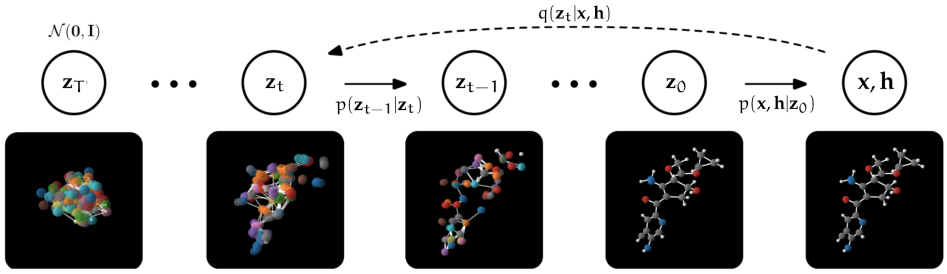

Figure 21: Overview of the Equivariant Diffusion Model. To generate molecules, coordinates $\mathbf{x}$ and features $\mathbf{h}$ are generated by denoising variables $\mathbf{z}(t)$ starting from standard normal noise $\mathbf{z}(T)$. This is achieved by sampling from the distributions $p(\mathbf{z}(t-1)|\mathbf{z}(t))$ iteratively. To train the model, noise is added to a datapoint $\mathbf{x}, \mathbf{h}$ using $q(\mathbf{z}(t)|\mathbf{x}, \mathbf{h})$ for the step t of interest, which the network then learns to denoise.

7.2 ACKNOWLEDGEMENT OF CONTRIBUTIONS

The work (Hoogeboom et al., 2022), upon which this chapter is based, was equally contributed to by Emiel Hoogeboom, Victor Garcia Satorras, and Clement Vignac. These three authors contributed significantly to the majority of the ideas, conducted the experiments, and wrote the paper. There was a division of tasks in certain sections: Victor was responsible for the conditional model experiments and EGNN integration. Emiel and Clement carried out the experiment on the Geom-drugs dataset. A summary of the results from the Geom-drugs experiment is still included in this thesis with the co-authors' consent, to exemplify the model's scalability. Additionally, the original paper contains a probabilistic analysis that enables likelihood computation with discrete and continuous features, which was formalized by Emiel. As this analysis falls outside of the scope of this thesis, it has been omitted but can be found in Section 3 of the paper. Finally, all remaining experiments and model sections, such as the primary model design and the QM9 experiment, were significantly contributed to by all authors.

7.3 BACKGROUND

7.3.1 *Notation Clarification*

In this section we adhere to the same notation used in previous Chapter 6 in which we represent both vectors and matrices with lower lowercase letters. For example, $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_M) \in \mathbb{R}^{M \times n}$. This differs from what was stated in the Background Section 2.1.

7.3.2 *Diffusion Models*

Diffusion models learn distributions by modeling the reverse of a diffusion process, termed a *denoising* process. Given a data point $\mathbf{x}$, the diffusion (forward)

process is defined as a Markov chain of T steps that gradually adds Gaussian noise to the sample $\mathbf{x}$. Because the sum of Gaussian distributed variables remains Gaussian, we can define the probability distribution of an intermediate noisy step $\mathbf{z}(t)$ for $t \in [0, \dots, T]$ as the following multivariate normal distribution:

$$q(\mathbf{z}(t)|\mathbf{x}) = \mathcal{N}(\mathbf{z}(t)|\alpha_t \mathbf{x}, \sigma_t^2 \mathbf{I}), \tag{57}$$

where $\alpha_t \in \mathbb{R}^+$ and $\sigma_t \in \mathbb{R}^+$ control how much signal is retained and how much noise is added respectively. In general, α_t is modelled by a function that smoothly transitions from $\alpha_0 \approx 1$ towards $\alpha_T \approx 0$. A special case of noising process is the variance preserving process (Sohl-Dickstein et al., 2015; Ho et al., 2020), for which $\alpha_t = \sqrt{1 - \sigma_t^2}$. Following Kingma et al. (2021), we define the signal to noise ratio $\text{SNR}(t) = \alpha_t^2 / \sigma_t^2$, which simplifies notations. This diffusion process is Markov and can be equivalently written with transition distributions as:

$$q(\mathbf{z}(t)|\mathbf{z}(s)) = \mathcal{N}(\mathbf{z}(t)|\alpha_{t|s}\mathbf{z}(s),\ \sigma_{t|s}^2 \mathbf{I}), \tag{58}$$

for any $t > s$ with $\alpha_{t|s} = \alpha_t / \alpha_s$ and $\sigma_{t|s}^2 = \sigma_t^2 - \alpha_{t|s}^2 \sigma_s^2$. The entire noising process is then written as:

$$q(\mathbf{z}(0), \mathbf{z}(1), \dots, \mathbf{z}(T)|\mathbf{x}) = q(\mathbf{z}(0)|\mathbf{x}) \prod_{t=1}^{T} q(\mathbf{z}(t)|\mathbf{z}(t-1)). \tag{59}$$

The posterior of the transitions conditioned on $\mathbf{x}$ gives the inverse of the noising process, the *true denoising process*. It is also normal and given by:

$$q(\mathbf{z}(s)|\mathbf{x}, \mathbf{z}(t)) = \mathcal{N}(\mathbf{z}(s)|\boldsymbol{\mu}_{t\to s}(\mathbf{x}, \mathbf{z}(t)), \sigma_{t\to s}^2 \mathbf{I}), \tag{60}$$

where the definitions for $\boldsymbol{\mu}_{t\to s}(\mathbf{x}, \mathbf{z}(t))$ and $\sigma_{t\to s}$ can be analytically obtained as

$$\boldsymbol{\mu}_{t\to s}(\mathbf{x}, \mathbf{z}(t)) = \frac{\alpha_{t|s}\sigma_s^2}{\sigma_t^2}\mathbf{z}(t) + \frac{\alpha_s \sigma_{t|s}^2}{\sigma_t^2}\mathbf{x} \quad \text{and} \quad \sigma_{t\to s} = \frac{\sigma_{t|s}\sigma_s}{\sigma_t}.$$

7.3.3 *The Generative Denoising Process*

In contrast to other generative models, in diffusion models, the generative process is defined with respect to the *true denoising process*. The variable $\mathbf{x}$, which is unknown to the generative process, is replaced by an approximation $\hat{\mathbf{x}} = \phi(\mathbf{z}(t), t)$ given by a neural network ϕ. Then the generative transition distribution $p(\mathbf{z}(s)|\mathbf{z}(t))$ is chosen to be $q(\mathbf{z}(s)|\hat{\mathbf{x}}(\mathbf{z}(t), t), \mathbf{z}(t))$. Similarly to Eq. 60, it can be expressed using the approximation $\hat{\mathbf{x}}$ as:

$$p(\mathbf{z}(s)|\mathbf{z}(t)) = \mathcal{N}(\mathbf{z}(s)|\boldsymbol{\mu}_{t\to s}(\hat{\mathbf{x}}, \mathbf{z}(t)), \sigma_{t\to s}^2 \mathbf{I}). \tag{61}$$

With the choice $s = t-1$, a variational lower bound on the log-likelihood of $\mathbf{x}$ given the generative model is given by:

$$\log p(\mathbf{x}) \geqslant \mathcal{L}_0 + \mathcal{L}_{\text{base}} + \sum_{t=1}^{T} \mathcal{L}_t, \tag{62}$$

where $\mathcal{L}_0 = \log p(\mathbf{x}|\mathbf{z}(0))$ models the likelihood of the data given $\mathbf{z}(0)$, $\mathcal{L}_{\text{base}} = -\text{KL}(q(\mathbf{z}(T)|\mathbf{x})|p(\mathbf{z}(T)))$ models the distance between a standard normal distribution and the final latent variable $q(\mathbf{z}(T)|\mathbf{x})$, and

$$\mathcal{L}_t = -\text{KL}(q(\mathbf{z}(s)|\mathbf{x}, \mathbf{z}(t))|p(\mathbf{z}(s)|\mathbf{z}(t))) \quad \text{for } t = 1, \ldots, T.$$

While in this formulation the neural network directly predicts $\hat{\mathbf{x}}$, Ho et al. (2020) found that optimization is easier when predicting the Gaussian noise instead. Intuitively, the network is trying to predict which part of the observation $\mathbf{z}(t)$ is noise originating from the diffusion process, and which part corresponds to the underlying data point $\mathbf{x}$. Specifically, if $\mathbf{z}(t) = \alpha_t \mathbf{x} + \sigma_t \boldsymbol{\epsilon}$, then the neural network ϕ outputs $\hat{\boldsymbol{\epsilon}} = \phi(\mathbf{z}(t), t)$, so that:

$$\hat{\mathbf{x}} = (1/\alpha_t)\ \mathbf{z}(t) - (\sigma_t/\alpha_t)\ \hat{\boldsymbol{\epsilon}} \tag{63}$$

As shown in (Kingma et al., 2021), with this parametrization $\mathcal{L}_t$ simplifies to:

$$\mathcal{L}_t = \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\frac{1}{2} (1 - \text{SNR}(t-1)/\text{SNR}(t)) \|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}\|^2 \right] \tag{64}$$

In practice the term $\mathcal{L}_{\text{base}}$ is close to zero when the noising schedule is defined in such a way that $\alpha_T \approx 0$. Furthermore, if $\alpha_0 \approx 1$ *and* $\mathbf{x}$ is discrete, then $\mathcal{L}_0$ is close to zero as well.

7.3.4 *E(n) Equivariant Graph Neural Networks (EGNNs)*

In this chapter, we will model the dynamics ϕ of the diffusion model, introduced in Section 7.3.3, using the EGNN (Satorras et al., 2021b) from Chapter 5. However, we introduce a further modification to the network that slightly improves its performance. Given a fully connected graph $\mathcal{G}$ with nodes $v_i \in \mathcal{V}$, each node v_i is endowed with coordinates $\mathbf{x}_i \in \mathbb{R}^3$ as well as features $\mathbf{h}_i \in \mathbb{R}^d$. We define this new EGNN layer as:

$$\mathbf{m}_{ij} = \phi_e \left(\mathbf{h}_i^l, \mathbf{h}_j^l, d_{ij}^2, \mathbf{a}_{ij} \right), \ \mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \sum_{j \neq i} \tilde{e}_{ij} \mathbf{m}_{ij}),$$

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \sum_{j \neq i} \frac{\mathbf{x}_i^l - \mathbf{x}_j^l}{d_{ij} + 1} \phi_x \left(\mathbf{h}_i^{l+1}, \mathbf{h}_j^{l+1}, d_{ij}^2, \mathbf{a}_{ij} \right), \tag{65}$$

where l indexes the layer, and $d_{ij} = \|\mathbf{x}_i^l - \mathbf{x}_j^l\|_2$ is the euclidean distance between nodes (v_i, v_j). Note that the difference $(\mathbf{x}_i^l - \mathbf{x}_j^l)$ in Equation 65 is normalized by $(d_{ij} + 1)$ as we previously did in 6 for improved stability. Additionally, in this chapter, we modify the coordinate update function inputs ϕ_x, which now takes as arguments $(\mathbf{h}_i^{l+1}, \mathbf{h}_j^{l+1}, d_{ij}^2, \mathbf{a}_{ij})$. In chapters 5 and 6, this function took as an argument the output of the function ϕ_e. This modification results in less parameter sharing and has shown slightly better performance.

The details on how the EGNN is integrated within the diffusion model are explained in the method Section 7.4. All learnable components (ϕ_e, ϕ_h, ϕ_x and ϕ_{inf}) are parametrized by fully connected neural networks specified in Appendix D.1.1.

7.4 EDM: E(3) EQUIVARIANT DIFFUSION MODEL

In this section, we describe the EDM, an E(3) Equivariant Diffusion Model. EDM defines a noising process for both node positions and features, and it uses the E(n) Equivariant Graph Neural Network to learn the generative *denoising* process.

7.4.1 *The Diffusion Process*

We first define an equivariant diffusion process for coordinates $\mathbf{x}_i$ with atom features $\mathbf{h}_i$ that adds noise to the data. Recall that we consider a set of points $\{(\mathbf{x}_i, \mathbf{h}_i)\}_{i=1,\dots,M}$, where each node has associated to it a coordinate representation $\mathbf{x}_i \in \mathbb{R}^n$ and an attribute vector $\mathbf{h}_i \in \mathbb{R}^{nf}$. Let $[\cdot, \cdot]$ denote a concatenation. We define the equivariant noising process on latent variables $\mathbf{z}(t) = [\mathbf{z}(t)^{(x)}, \mathbf{z}(t)^{(h)}]$ as:

$$q(\mathbf{z}(t)|\mathbf{x}, \mathbf{h}) = \mathcal{N}_{xh}(\mathbf{z}(t)|\alpha_t[\mathbf{x}, \mathbf{h}], \sigma_t^2\mathbf{I}) \tag{66}$$

for $t = 1, \dots, T$ where $\mathcal{N}_{xh}$ is concise notation for the product of two distributions, one for the noised coordinates $\mathcal{N}_x$ and another for the noised features $\mathcal{N}$ given by:

$$\mathcal{N}_x(\mathbf{z}(t)^{(x)}|\alpha_t\mathbf{x}, \sigma_t^2\mathbf{I}) \cdot \mathcal{N}(\mathbf{z}(t)^{(h)}|\alpha_t\mathbf{h}, \sigma_t^2\mathbf{I}) \tag{67}$$

These equations correspond to Equation 57 in a standard diffusion model. Also, a slight abuse of notation is used to aid readability: technically $\mathbf{x}$, $\mathbf{h}$, $\mathbf{z}(t)$ are two-dimensional variables with an axis for the point identifier and an axis for the features. However, in the distributions they are treated as if flattened to a vector.

As explained in the previous chapter, Section 6.5.3, it is impossible to have a non-zero distribution that is invariant to translations, as it cannot integrate to one. However, one can use distributions on the linear subspace where the center of gravity is always zero. Following (Xu et al., 2022), which showed that such a

Algorithm 1 Optimizing EDM

Input: Data point $\mathbf{x}$, neural network ϕ
Sample $t \sim \mathcal{U}(0, \dots, T)$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Subtract center of gravity from $\boldsymbol{\epsilon}^{(x)}$ in $\boldsymbol{\epsilon} = [\boldsymbol{\epsilon}^{(x)}, \boldsymbol{\epsilon}^{(h)}]$
Compute $\mathbf{z}(t) = \alpha_t[\mathbf{x}, \mathbf{h}] + \sigma_t \epsilon$
Minimize $\|\boldsymbol{\epsilon} - \phi(\mathbf{z}(t), t)\|^2$

Algorithm 2 Sampling from EDM

Sample $\mathbf{z}(t) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for t in T, $T-1, \dots, 1$ where $s = t-1$ **do**
  Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Subtract center of gravity from $\boldsymbol{\epsilon}^{(x)}$ in $\boldsymbol{\epsilon} = [\boldsymbol{\epsilon}^{(x)}, \boldsymbol{\epsilon}^{(h)}]$
  $\mathbf{z}_s = \frac{1}{\alpha_{t|s}}\mathbf{z}(t) - \frac{\sigma^2_{t|s}}{\alpha_{t|s}\sigma_t} \cdot \phi(\mathbf{z}(t), t) + \sigma_{t \to s} \cdot \boldsymbol{\epsilon}$
end for
Sample $\mathbf{x}, \mathbf{h} \sim p(\mathbf{x}, \mathbf{h} | \mathbf{z}(0))$

linear subspace can be consistently used in diffusion, $\mathcal{N}_x$ is defined as a normal distribution on the subspace defined by $\sum_i \mathbf{x}_i = \mathbf{0}$. Recalling the definition from previous Chapter 6.5.4, $\mathcal{N}_x\left(\mathbf{x} \mid \boldsymbol{\mu}, \sigma^2\mathbf{I}\right) = (\sqrt{2\pi}\sigma)^{-(M-1)\cdot n} \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \boldsymbol{\mu}\|^2\right)$.

Since the features $\mathbf{h}$ are invariant to E(n) transformations, the noise distribution for these features can be the conventional normal distribution $\mathcal{N}$.

7.4.2 *The Generative Denoising Process*

To define the generative process, the noise posteriors $q(\mathbf{z}(s)|\mathbf{x}, \mathbf{h}, \mathbf{z}(t))$ of Equation 66 can be used in the same fashion as in Equation 60 by replacing the data variables $\mathbf{x}, \mathbf{h}$ by neural network approximations $\hat{\mathbf{x}}, \hat{\mathbf{h}}$:

$$p(\mathbf{z}(s)|\mathbf{z}(t)) = \mathcal{N}_{xh}(\mathbf{z}(s)|\boldsymbol{\mu}_{t\to s}([\hat{\mathbf{x}}, \hat{\mathbf{h}}], \mathbf{z}(t)), \sigma^2_{t\to s}\mathbf{I}) \tag{68}$$

where $\hat{\mathbf{x}}, \hat{\mathbf{h}}$ depend on $\mathbf{z}(t), t$ and the neural network ϕ. As conventional in modern diffusion models, we use the noise parametrization to obtain $\hat{\mathbf{x}}, \hat{\mathbf{h}}$. Instead of directly predicting them, the network ϕ outputs $\hat{\boldsymbol{\epsilon}} = [\hat{\boldsymbol{\epsilon}}^{(x)}, \hat{\boldsymbol{\epsilon}}^{(h)}]$ which is then used to compute:

$$[\hat{\mathbf{x}}, \hat{\mathbf{h}}] = \mathbf{z}(t)/\alpha_t - \hat{\boldsymbol{\epsilon}}_t \cdot \sigma_t/\alpha_t \tag{69}$$

If $\hat{\boldsymbol{\epsilon}}_t$ is computed by an equivariant function ϕ then the denoising distribution in Equation 68 is equivariant. To see this, observe that rotating $\mathbf{z}(t)$ to $\mathbf{R}\mathbf{z}(t)$ gives $\mathbf{R}\hat{\boldsymbol{\epsilon}}_t = \phi(\mathbf{R}\mathbf{z}(t), t)$. Furthermore, the mean of the denoising equation rotates $\mathbf{R}\hat{\mathbf{x}} = \mathbf{R}\mathbf{z}(t)^{(x)}/\alpha_t - \mathbf{R}\hat{\boldsymbol{\epsilon}}_t^{(x)}\sigma_t/\alpha_t$ and since the noise is isotropic, the distribution is equivariant as desired.

To sample from the model, one first samples $\mathbf{z}(t) \sim \mathcal{N}_{xh}(\mathbf{0}, \mathbf{I})$ and then iteratively samples $\mathbf{z}(t-1) \sim p(\mathbf{z}(t-1)|\mathbf{z}(t))$ for $t = T, \dots, 1$ and then finally samples $\mathbf{x}, \mathbf{h} \sim p(\mathbf{x}, \mathbf{h}|\mathbf{z}(0))$, as described in Algorithm 2.

7.4.3 *Optimization Objective*

Recall that the likelihood term of this model is $\mathcal{L}_t = -\mathrm{KL}(q(\mathbf{z}(s)|\mathbf{x}, \mathbf{z}(t))\|p(\mathbf{z}(s)|\mathbf{z}(t)))$. Analogous to Equation 64, in this parametrization the term simplifies to:

$$\mathcal{L}_t = \mathbb{E}_{\boldsymbol{\epsilon}_t \sim \mathcal{N}_{xh}(0,\, \mathbf{I})}\big[\frac{1}{2}\, w(t)\, \|\boldsymbol{\epsilon}_t - \hat{\boldsymbol{\epsilon}}_t\|^2\big], \tag{70}$$

where $w(t) = (1 - \mathrm{SNR}(t-1)/\mathrm{SNR}(t))$ and $\hat{\boldsymbol{\epsilon}}_t = \phi(\mathbf{z}(t), t)$. This is convenient: even though parts of the distribution of $\mathcal{N}_{xh}$ operate on a subspace, the simplification in Equation 64 also holds here, and can be computed for all components belonging to $\mathbf{x}$ and $\mathbf{h}$ at once. There are three reason why this simplification remains true: firstly, $\mathcal{N}_x$ and $\mathcal{N}$ within $\mathcal{N}_{xh}$ are independent, so the divergence can be separated into two divergences. Further, the KL divergence between the $\mathcal{N}_x$ components are still compatible with the standard KL equation for normal distributions, as they rely on a Euclidean distance (which is rotation invariant) and the distributions are isotropic with equal variance. Finally, because of the similarity in KL equations, the results can be combined again by concatenating the components in $\mathbf{x}$ and $\mathbf{h}$. An overview of the optimization procedure is given in Algorithm 1.

Following (Ho et al., 2020) during training we set $w(t) = 1$ as it stabilizes training and it is known to improve sample quality for images. Experimentally we also found this to hold true for molecules: even when evaluating the probabilistic variational objective for which $w(t) = (1 - \mathrm{SNR}(t-1)/\mathrm{SNR}(t))$, the model trained with $w(t) = 1$ outperformed models trained with the variational $w(t)$.

In summary, we have defined a diffusion process, a denoising model and an optimization objective between them. To further specify our model, we need to define the neural network ϕ that is used within the denoising model.

7.4.4 *The Dynamics*

We learn the E(n) equivariant dynamics function $[\hat{\boldsymbol{\epsilon}}_t^{(x)}, \hat{\boldsymbol{\epsilon}}_t^{(h)}] = \phi(\mathbf{z}(t)^{(x)}, \mathbf{z}(t)^{(h)}, t)$ of the diffusion model using the equivariant network EGNN introduced in Section 7.3.4 in the following way:

$$\hat{\boldsymbol{\epsilon}}_t^{(x)}, \hat{\boldsymbol{\epsilon}}_t^{(h)} = \mathrm{EGNN}(\mathbf{z}(t)^{(x)}, [\mathbf{z}(t)^{(h)}, t/T]) - [\mathbf{z}(t)^{(x)}, \mathbf{0}]$$

Notice that we simply input $\mathbf{z}(t)^{(x)}, \mathbf{z}(t)^{(h)}$ to the EGNN with the only difference that t/T is concatenated to the node features. The estimated noise $\hat{\epsilon}_t^{(x)}$ is given by the output of the EGNN from which the input coordinates $\mathbf{z}(t)^{(x)}$ are removed. Importantly, since the outputs have to lie on a zero center of gravity subspace, the component $\hat{\epsilon}_t^{(x)}$ is projected down by subtracting its center of gravity. This then satisfies the rotational and reflection equivariance on $\hat{\mathbf{x}}$ with the parametrization in Equation 69. Additional details about the dynamics are provided in Appendix D.1.1

7.4.5 *Categorical Features*

For categorical features such as atom type, the aforementioned integer representation is unnatural and can introduce bias. Instead of using integers for these features, we operate directly on a one-hot representation. Suppose $\mathbf{h}$ is an array with values representing categories in $c_1, ..., c_d$, such as atom types. Then, $\mathbf{h}$ is encoded with a one-hot function, $\mathbf{h} \mapsto \mathbf{h}\text{onehot}$, such that $\mathbf{h}\text{onehot}_{i,j} = \mathbb{1}h_i = c_j$. The noising process over $\mathbf{z}(t)^{(h)}$ can then be directly defined using the one-hot representation $\mathbf{h}^{\text{onehot}}$, similar to its definition for integer values, i.e., $q(\mathbf{z}(t)^{(h)}|\mathbf{h}) = \mathcal{N}(\mathbf{z}(t)^{(h)}|\alpha_t \mathbf{h}^{\text{onehot}}, \sigma_t^2 \mathbf{I})$. The only difference is that $\mathbf{z}(t)^{(h)}$ has an additional dimension axis with a size equal to the number of categories. Since the data is discrete and the noising process is assumed to be well defined, we can define probability parameters $\mathbf{p}$ to be proportional to the normal distribution integrated from $1 - \frac{1}{2}$ to $1 + \frac{1}{2}$ using the same reasoning as for integer data. Intuitively, when a small amount of noise is sampled and added to the one-hot representation, the value corresponding to the active class will almost certainly fall between $1 - \frac{1}{2}$ and $1 + \frac{1}{2}$:

$$p(\mathbf{h}|\mathbf{z}(0)^{(h)}) = \mathcal{C}(\mathbf{h}|\mathbf{p}), \mathbf{p} \propto \int_{1-\frac{1}{2}}^{1+\frac{1}{2}} \mathcal{N}(\mathbf{u}|\mathbf{z}(0)^{(h)}, \sigma_0) d\mathbf{u}$$

Where $\mathbf{p}$ is normalized to sum to one and $\mathcal{C}$ represents a categorical distribution. In practice, this distribution will almost certainly equal one for values of $\mathbf{z}(0)^{(h)}$ that were sampled from the diffusion process given $\mathbf{h}$.

7.4.6 *Scaling Features*

Coordinates and atom types each represent different physical quantities. Consequently, we can establish a relative scaling among them. While normalizing these features simplifies their processing for the neural network, relative scaling has a more profound impact on the model. For instance, when the features $\mathbf{h}$ are defined on a smaller scale than the coordinates $\mathbf{x}$, the denoising process tends to initially determine rough positions and only decide on the atom types afterwards. Our empirical findings indicate that defining the input to our EDM model

as $[\mathbf{x}, 0.25\ \mathbf{h}^{\text{onehot}}]$ significantly enhances performance compared to non-scaled inputs

7.4.7 *Modelling the Number of Atoms*

In the above sections we have considered the number of atoms M to be known beforehand. To accommodate molecules of varying sizes, we calculate the categorical distribution $p(M)$ of molecule sizes based on the training set. To sample from the model $p(\mathbf{x}, \mathbf{h}, M)$, we first sample $M \sim p(M)$, and then sample $\mathbf{x}, \mathbf{h} \sim p(\mathbf{x}, \mathbf{h}|M)$ from the EDM. Although this conditioning on M is often omitted for clarity, it remains an important part of the generative process.

7.4.8 *Conditional Generation*

In this section we describe a straightforward extension to the proposed method to do conditional generation $\mathbf{x}, \mathbf{h} \sim p(\mathbf{x}, \mathbf{h}|c)$ given some desired property c. We can define the optimization lower bound for the conditional case as $\log p(\mathbf{x}, \mathbf{h}|c) \geqslant \mathcal{L}_{c,0} + \mathcal{L}_{c,\text{base}} + \sum_{t=1}^{T} \mathcal{L}_{c,t}$, where the different $\mathcal{L}_{c,t}$ for $1 \leqslant t < T - 1$ are defined similarly to Equation 70, with the important difference that the function $\hat{\epsilon}_t = \phi(\mathbf{z}(t), [t, c])$ takes as additional input a property c which is concatenated to the nodes features. Given a trained conditional model we define the generative process by first sampling the number of nodes M and a property value c from a parametrized distribution $c, M \sim p(c, M)$ defined in Appendix D.5. Next, we can generate molecules $\mathbf{x}, \mathbf{h}$ given c, M using our Conditional EDM $\mathbf{x}, \mathbf{h} \sim p(\mathbf{x}, \mathbf{h}|c, M)$.

7.5 RELATED WORK

Diffusion models (Sohl-Dickstein et al., 2015) are generative models that have been recently connected to score-based methods through denoising diffusion models (Song and Ermon, 2019; Ho et al., 2020). This new family of generative models has proven to be very effective in generating data such as images (Ho et al., 2020; Nichol and Dhariwal, 2021).

In molecule generation, there are some recent methods that generate molecules directly in their 3D form. G-Schnet (Gebauer et al., 2019) defines an autoregressive distribution from which atoms can be iteratively sampled, and can be used for targeted generation (Gebauer et al., 2021). E-NF (Satorras et al., 2021a, Chapter 6) defines an equivariant normalizing flow via a differential equation. Instead of integrating a differential equation, our method learns to denoise a diffusion process, which scales more favourably during training.

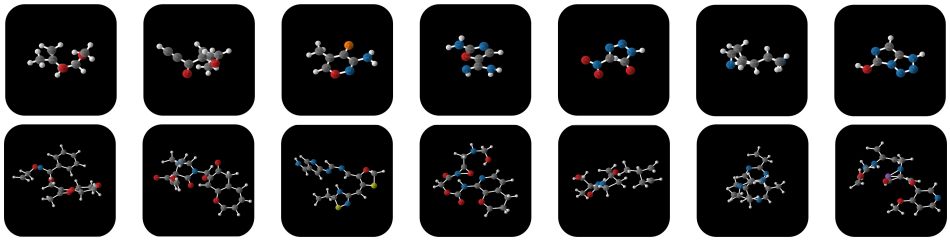

Figure 22: Selection of samples generated by the denoising process of our EDM trained on QM9 (up) and GEOM-DRUGS (down).

A related branch of literature is concerned with predicting coordinates from molecular graphs, referred to as the conformation. Examples of such methods utilize conditional VAEs (Simm and Hernández-Lobato, 2019), Wasserstein GANs (Hoffmann and Noé, 2019), and normalizing flows (Noé et al., 2019), with adaptations for Euclidean symmetries in (Köhler et al., 2020b; Xu et al., 2021b; Simm et al., 2021; Ganea et al., 2021; Guan et al., 2021) resulting in performance improvements. In recent works (Shi et al., 2021; Luo et al., 2021; Xu et al., 2022), it was shown that score-based and diffusion models are effective at coordinate prediction, especially when the underlying neural network respects the symmetries of the data. Our work can be seen as an extension of these methods that incorporates discrete atom features, and further derives the equations required for log-likelihood computation. In the context of diffusion for discrete variables, unrelated to molecule modelling, discrete diffusion processes have been proposed (Sohl-Dickstein et al., 2015; Hoogeboom et al., 2021; Austin et al., 2021). However, for 3D molecule generation these would require a separate diffusion process for the discrete features and the continuous coordinates. Instead, we define a joint process for both of them.

Tangentially related, other methods generate molecules in graph representation. Examples include autoregressive methods such as (Liu et al., 2018; You et al., 2018; Liao et al., 2019), and one-shot approaches such as (Simonovsky and Komodakis, 2018; De Cao and Kipf, 2018; Bresson and Laurent, 2019; Kosiorek et al., 2020; Krawczuk et al., 2021). However, such methods do not provide conformer information, which is useful for many downstream tasks.

7.6 EXPERIMENTS

7.6.1 *Molecule Generation | QM9*

QM9 (Ramakrishnan et al., 2014) is a standard dataset that contains molecular properties and atom coordinates for 130k small molecules with up to 9 heavy atoms (29 atoms including hydrogens). In this experiment, we train EDM to unconditionally generate molecules with 3-dimensional coordinates, atom types (H, C, N, O, F), and integer-valued atom charges. We use the train/val/test partitions

Table 12: Neg. log-likelihood $-\log p(\mathbf{x}, \mathbf{h}, M)$, atom stability and molecule stability with standard deviations across 3 runs on QM9, each drawing 10000 samples from the model.

# Metrics	NLL	Atom stable (%)	Mol stable (%)
E-NF	-59.7	85.0	4.9
G-Schnet	N.A	95.7	68.1
GDM	-94.7	97.0	63.2
GDM-aug	-92.5	97.6	71.6
EDM (ours)	**−110.7**$\pm$1.5	**98.7**$\pm$0.1	**82.0**$\pm$0.4
Data		99.0	95.2

introduced in (Anderson et al., 2019), which consist of 100K/18K/13K samples respectively for each partition.

METRICS: Following (Satorras et al., 2021a), we use the distance between pairs of atoms and the atom types to predict bond types (single, double, triple or none). We then measure atom stability (the proportion of atoms that have the right valency) and molecule stability (the proportion of generated molecules for which all atoms are stable).

BASELINES: We compare EDM to two existing E(3) equivariant models: G-Schnet (Gebauer et al., 2019) and Equivariant Normalizing Flows (E-NF) (Satorras et al., 2021a). For G-Schnet, we extracted 10000 samples from the publicly available code to run the analysis. To demonstrate the benefits of equivariance, we also perform an ablation study and run a non-equivariant variation of our method that we call Graph Diffusion Models (GDM). The Graph diffusion model is run with the same configuration as our method, except that the EGNN is replaced by a non-equivariant graph network defined in Appendix D.2. We also experiment with GDM-aug, where the GDM model is trained on data augmented with random rotations. All models use 9 layers, 256t features per layer, and SiLU activations. They are trained using Adam with a batch size of 64 and a learning rate of 10^{-4}.

RESULTS: Results are reported in Table 12. Our method outperforms previous methods (E-NF and G-Schnet), as well as its non-equivariant counterparts on all metrics. It is interesting to note that the negative log-likelihood of the EDM is much lower than other models, which indicates that it is able to create sharper peaks in the model distribution.

Further, EDMs are compared to one-shot graph-based molecule generation models that do not operate on 3D coordinates: GraphVAE (Simonovsky and Komodakis, 2018), GraphTransformerVAE (Mitton et al., 2021), and Set2GraphVAE (Vignac and Frossard, 2021). For G-Schnet and EDM, the bonds are directly

Table 13: Validity and uniqueness over 10000 molecules with standard deviation across 3 runs. Results marked (*) are not directly comparable, as they do not use 3D coordinates to derive bonds. H: model hydrogens explicitly

Method	H	Valid (%)	Valid and Unique (%)
Graph VAE (*)		55.7	42.3
GTVAE (*)		74.6	16.8
Set2GraphVAE (*)		59.9±1.7	56.2±1.4
EDM (ours)		**97.5**±0.2	**94.3**±0.2
E-NF	✓	40.2	39.4
G-Schnet	✓	85.5	80.3
GDM-aug	✓	90.4	89.5
EDM (ours)	✓	**91.9**±0.5	**90.7**±0.6
Data	✓	97.7	97.7

derived from the distance between atoms. We report validity (as measured by RDKit) and uniqueness of the generated compounds. Following (Vignac and Frossard, 2021) novelty is not included here. For a discussion on the issues with the novelty metric, see Appendix D.2. As can be seen in Table 13, the EDM is able to generate a very high rate of valid and unique molecules. This is impressive since the 3D models are at a disadvantage in this metric, as the rules to derive bonds are very strict. Interestingly, even when including hydrogen atoms in the model, the performance of the EDM does not deteriorate much. A possible explanation is that the equivariant diffusion model scales effectively and learn very precise distributions, as evidenced by the low negative log-likelihood.

7.6.2 *Conditional Molecule Generation*

In this section, we aim to generate molecules targeting some desired properties. This can be of interest towards the process of drug discovery where we need to obtain molecules that satisfy specific properties. We train our conditional diffusion model from Section 7.4.8 in QM9 conditioning the generation on properties α, gap, homo, lumo, μ and C_ν described in more detail in Appendix D.5. In order to assess the quality of the generated molecules w.r.t. to their conditioned property, we use the property classifier network ϕ_c from Satorras et al. (2021b). We split the QM9 training partition into two halves D_a, D_b of 50K samples each. The classifier ϕ_c is trained on the first half D_a, while the Conditional EDM is trained on the second half D_b. Then, ϕ_c is evaluated on the EDM conditionally generated samples. We also report the loss of ϕ_c on D_b as a lower bound named "QM9 (L-bound)". The better EDM approximates D_b the smaller the gap between "EDM" and "QM9 (L-bound)". Further implementation details are reported in Appendix D.5.

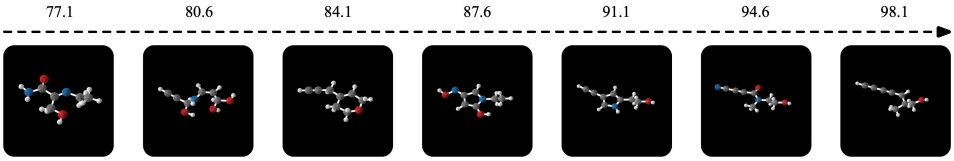

Figure 23: Generated molecules by our Conditional EDM when interpolating among different Polarizability α values with the same reparametrization noise ϵ. Each α value is provided on top of each image.

Table 14: Mean Absolute Error for molecular property prediction by a EGNN classifier ϕ_c on a QM9 subset, EDM generated samples and two different baselines "Naive (U-bounds)" and "# Atoms".

Task	α	$\Delta\varepsilon$	ε_{HOMO}	ε_{LUMO}	μ	C_ν
Units	$Bohr^3$	meV	meV	meV	D	$\frac{cal}{mol}K$
Naive (U-bound)	9.01	1470	645	1457	1.616	6.857
#Atoms	3.86	866	426	813	1.053	1.971
EDM	2.76	655	356	584	1.111	1.101
QM9 (L-bound)	0.10	64	39	36	0.043	0.040

BASELINES: We provide two baselines in which molecules are to some extent agnostic to their respective property c. In the first baseline we simply remove any relation between molecule and property by shuffling the property labels in D_b and then evaluating ϕ_c on it. We name this setting "Naive (Upper-Bound)". The second baseline named "#Atoms" predicts the molecular properties in D_b by only using the number of atoms in the molecule. If "EDM" overcomes "Naive (Upper-Bound)" it should be able to incorporate conditional property information into the generated molecules. If it overcomes "#Atoms" it should be able to incorporate it into the molecular structure beyond the number of atoms.

RESULTS (QUANTITATIVE): Results are reported in Table 14. EDM outperforms both "Naive (U-bound)" and "#Atoms" baselines in all properties (except μ) indicating that it is able to incorporate property information into the generated molecules beyond the number of atoms for most properties. However, we can see there is still room for improvement by looking at the gap between "EDM" and "QM9 (L-bound)".

RESULTS (QUALITATIVE): In Figure 23, we interpolate the conditional generation among different Polarizability values α while keeping the noise ϵ fixed. The Polarizability is the tendency of a molecule to acquire an electric dipole moment when subject to an external electric field. We can expect less isometrically shaped molecules for large α values. This is the obtained behavior in Figure 23 – we show that this behavior is consistent across different runs in Appendix D.5.

Table 15: Neg. log-likelihood, atom stability and Wasserstein distance between generated and training set energy distributions.

# Metrics	NLL	Atom stability (%)	$\mathcal{W}$
GDM	− 14.2	75.0	3.32
GDM-aug	− 58.3	77.7	4.26
EDM	**−137.1**	**81.3**	**1.41**
Data		86.5	0.0

7.6.3 *GEOM-Drugs*

While QM9 features only small molecules, GEOM (Axelrod and Gomez-Bombarelli, 2020) is a larger scale dataset of molecular conformers. It features 430,000 molecules with up to 181 atoms and an average of 44.4 atoms. For each molecule, many conformers are provided along with their energy. From this dataset, we retain the 30 lowest energy conformations for each molecule. The models are trained to generate the 3D positions and atom types of these molecules. All models use 4 layers, 256 features per layer, and are trained using Adam with a batch size of 64 and a learning rate of 10^{-4}.

Since molecules in this dataset are larger and have more complex structures, predicting the bond types using the atom types and the distance between atoms with lookup tables results in more errors than on QM9. For this reason, we only report the atom stability, which measures 86.5% stable atoms on the dataset. Intuitively, this metric describes the percentage of atoms that have bonds within typical ranges – ideally, generative models should generate a comparable number of stable atoms. In Table 15 we can see that the EDM outperforms its non-equivariant counterparts on all metrics. In particular, EDM is able to capture the energy distribution well, as seen in the histograms in Appendix D.2.

7.7 CONCLUSIONS

In this chapter, we introduced an E(3) Equivariant Diffusion Model (EDM) for 3D molecule generation. This model builds upon the E(n) Equivariant Graph Neural Networks from Chapter 5 and addresses the primary scalability limitations found in Equivariant Normalizing Flows from Chapter 6. To the best of our knowledge, EDM is the first diffusion model for 3D molecule generation, establishing a new benchmark in literature. We demonstrate that our model provides superior results to previous molecular generative models and scales to larger molecular data such as GEOM-DRUGS.

Deep learning generative methods hold potential to impact the field of molecular sciences. Particularly, subsequent to this work, numerous diffusion models for molecule generation are influencing application such as molecule-ligand interactions (Igashov et al., 2022; Schneuing et al., 2022), drug discovery (Isert et al., 2023) and material design (Zeni et al., 2023).

8

CONCLUSIONS

8.1 SUMMARY OF CONCLUSIONS

The main contribution of this thesis has been the enhancement of Graph Neural Networks (GNNs) through the incorporation of physics-based inductive biases, yielding significant improvements across a range of applications, with a particular emphasis on molecular modeling.

The initial part of the thesis explores the combination of recent advances in GNNs with more traditional methods of graphical inference. The result is a hybrid algorithm that benefits from both the inductive bias of graphical models and the high expressivity in data-driven inference of neural networks. The proposed hybrid algorithm demonstrates better performance than its individual components when run in isolation on a variety of tasks such as estimating chaotic trajectories.

Subsequently, the thesis presents a GNN designed to be equivariant to Euclidean transformations in a 3D space, a symmetry group with many important real-world applications such as molecular modeling. We named this model E(n) Equivariant Graph Neural Networks (EGNN). This EGNN is computationally efficient, simple to implement, and outperforms more expensive equivariant networks in molecular property prediction and dynamics forecasting. The efficiency and efficacy of the EGNN have made the network a valuable tool adopted in a wide variety of subsequent works in the context of molecular sciences, as mentioned in Section 1.3.

Further advancing the field of deep learning for molecular modeling, the thesis extends the EGNN to build E(n) Equivariant Generative Models, with a focus on molecular structure generation in three dimensions. The E(n) Equivariant Normalizing Flows (E-NFs) and the E(3) Equivariant Diffusion Model (EDM), the latter offering superior performance and scalability than previous literature. These

advancements in generative models have begun to influence applications in drug design and material sciences.

8.2 RESEARCH QUESTIONS

Having introduced our work, we can now attempt to answer the research questions:

Research Question 1: *How can we benefit from both the expressivity of GNNs and the generalization and data efficiency of graphical models?*

This thesis has demonstrated that it is possible to leverage the generalization capabilities of graphical models with the expressivity of GNNs. Through the design of a hybrid message-passing framework, we achieved better performance and generalization than pure graphical inference or pure GNN data-learned inference in a variety of tasks, such as estimating chaotic trajectories.

Research Question 2: *How can we build effective and yet fast graph neural networks that are equivariant to Euclidan transformations?*

By developing E(n) Equivariant Graph Neural Networks (EGNNs), we have demonstrated that it is possible to obtain an equivariant GNN that is both efficient and effective. This thesis has demonstrated the competitive performance of the EGNN in a variety of tasks such as molecular property prediction and the design of new equivariant generative models. Additionally, the EGNN's adoption in many subsequent works highlights its utility in addressing practical challenges in fields where Euclidean symmetries are inherent in the data.

Research Question 3: *How can we construct E(3) equivariant generative models and what advancements do they offer in the generation of molecular structures?*

This thesis made substantial contributions to the field of generative modeling in 3D Euclidean spaces by introducing E(n) Equivariant Normalizing Flows (E-NFs) and E(3) Equivariant Diffusion Models (EDMs). These methods represent a significant step forward in the generation of 3D molecular structures. The EDM, in particular, has set a new benchmark in 3D molecule generation, demonstrating better performance and scalability than previous literature. The impact of these models is beginning to influence applications in drug discovery and material design, holding potential to revolutionize molecular sciences.

8.3 LIMITATIONS AND FUTURE WORK

Despite the meaningful contributions presented in this thesis, they are just a small step towards the potential that machine learning has to disrupt a field such as molecular sciences.

Machine learning can potentially disrupt high-impact applications within molecular sciences, such as drug discovery, material design, and modeling chemical reactions, all of which can have a significant and hopefully positive impact in the real world. To reach a turning point where machine learning methods outperform classical chemistry methods, these models may need to become more accurate than the ones proposed in this thesis and more efficient than current methods used in computational chemistry.

For instance, if we could scale up a diffusion generative model, as presented in Chapter 7, to generalize across a wide variety of systems and provide highly accurate structures when conditioned on desired properties or structures, it would be an invaluable tool in tasks like material design, finding ligands given protein structures or catalysts for a desired chemical reaction. However, the methods presented in this thesis, such as Equivariant Diffusion Models (EDM) using E(n) Equivariant Graph Neural Networks, are limited to a small training domain, e.g., QM9 or Geom-Drugs.

While in domains like Computer Vision and Natural Language Processing, a solution to obtain more accurate machine learning models has been to simply scale up models with deeper networks and more training samples (Ramesh et al., 2021). In molecular sciences, we often face a slightly different paradigm regarding the nature of training samples. For instance, in Computer Vision, data is obtained from the real world and is accessible in huge quantities online. In contrast, in chemistry, the QM9 dataset used in the experiments of Chapters 5, 6, 7 was obtained with costly ab initio methods, such as Density Functional Theory computations based on quantum mechanics.

This reliance on classical methods for data generation to train machine learning models can result in a 'chicken and egg' problem, wherein the goal of the machine learning model is to approximate a similar distribution as the one we can already model with classical methods but more efficiently. However, this limitation also comes with a potential benefit: we have access to the equations that define these ab initio methods. Throughout this thesis, we have shown how to leverage prior knowledge, specifically, in Chapters 3 and 4 we have shown how to combine both physics equations with machine learning models, inferring more accurate trajectories of a Lorenz oscillator with fewer samples by leveraging the differential equations that define the trajectories. We have also leveraged Euclidean symmetry constraints in Chapters 5, 6, and 7.

This brings us to new research questions: Can we further leverage symmetries or prior knowledge present in physics, such as the energy conservation law, pre-defined interactions in force fields, or the anti-symmetry of electron interactions? Can we define better machine learning models that are efficient and accurate enough to significantly disrupt the field of molecular sciences through these newly defined inductive biases?

BIBLIOGRAPHY

Abanades, Brennan, Guy Georges, Alexander Bujotzek, and Charlotte M Deane (2022). "ABlooper: fast accurate antibody CDR loop structure prediction with accuracy estimation." In: *Bioinformatics* 38.7, pp. 1877–1880.

Abbeel, Pieter, Adam Coates, Michael Montemerlo, Andrew Y Ng, and Sebastian Thrun (2005). "Discriminative Training of Kalman Filters." In: *Robotics: Science and systems*. Vol. 2, p. 1.

AlQuraishi, Mohammed (2019). "AlphaFold at CASP13." In: *Bioinformatics* 35.22, pp. 4862–4865.

Anderson, Brandon, Truong Son Hy, and Risi Kondor (2019). "Cormorant: Covariant molecular neural networks." In: *Advances in neural information processing systems* 32.

Andrychowicz, Marcin, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas (2016). "Learning to learn by gradient descent by gradient descent." In: *Advances in Neural Information Processing Systems*, pp. 3981–3989.

Anstine, Dylan M and Olexandr Isayev (2023). "Generative Models as an Emerging Paradigm in the Chemical Sciences." In: *Journal of the American Chemical Society* 145.16, pp. 8736–8750.

Atz, Kenneth, Clemens Isert, Markus NA Böcker, José Jiménez-Luna, and Gisbert Schneider (2022). "Δ-Quantum machine-learning for medicinal chemistry." In: *Physical Chemistry Chemical Physics* 24.18, pp. 10775–10783.

Austin, Jacob, Daniel Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg (2021). "Structured denoising diffusion models in discrete state-spaces." In: *Advances in Neural Information Processing Systems* 34.

Axelrod, Simon and Rafael Gomez-Bombarelli (2020). "GEOM: Energy-annotated molecular conformations for property prediction and molecular generation." In: *arXiv preprint arXiv:2006.05531*.

Belanger, David, Bishan Yang, and Andrew McCallum (2017). "End-to-end learning for structured prediction energy networks." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 429–439.

Bengio, Yoshua (2017). "The consciousness prior." In: *arXiv preprint arXiv:1709.08568*.

Biloš, Marin and Stephan Günnemann (2020). "Equivariant normalizing flows for point processes and sets." In: *arXiv preprint arXiv:2010.03242*.

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer.

Bollobás, Béla (1998). "Random graphs." In: *Modern graph theory*. Springer, pp. 215–252.

Boyda, Denis, Gurtej Kanwar, Sébastien Racanière, Danilo Jimenez Rezende, Michael S Albergo, Kyle Cranmer, Daniel C Hackett, and Phiala E Shanahan (2021). "Sampling using SU (N) gauge equivariant flows." In: *Physical Review D* 103.7, p. 074504.

Brandstetter, Johannes, Rob Hesselink, Elise van der Pol, Erik J Bekkers, and Max Welling (2021). "Geometric and physical quantities improve e (3) equivariant message passing." In: *arXiv preprint arXiv:2110.02905*.

Braunstein, Alfredo and Riccardo Zecchina (2004). "Survey propagation as local equilibrium equations." In: *Journal of Statistical Mechanics: Theory and Experiment* 2004.06, P06007.

Bresson, Xavier and Thomas Laurent (2019). "A two-step graph convolutional decoder for molecule generation." In: *arXiv preprint arXiv:1906.03412*.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. (2020). "Language models are few-shot learners." In: *Advances in neural information processing systems* 33, pp. 1877–1901.

Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun (2013). "Spectral networks and locally connected networks on graphs." In: *arXiv preprint arXiv:1312.6203*.

Candès, Emmanuel J and Benjamin Recht (2009). "Exact matrix completion via convex optimization." In: *Foundations of Computational mathematics* 9.6, pp. 717–772.

Carlevaris-Bianco, Nicholas, Arash K Ushani, and Ryan M Eustice (2016). "University of Michigan North Campus long-term vision and lidar dataset." In: *The International Journal of Robotics Research* 35.9, pp. 1023–1035.

Chen, Changyou, Chunyuan Li, Liquan Chen, Wenlin Wang, Yunchen Pu, and Lawrence Carin (2018a). "Continuous-Time Flows for Efficient Inference and Density Estimation." In: *Proceedings of the 35th International Conference on Machine Learning, ICML*. Ed. by Jennifer G. Dy and Andreas Krause.

Chen, Changyou, Chunyuan Li, Liqun Chen, Wenlin Wang, Yunchen Pu, and Lawrence Carin (2017). "Continuous-time flows for deep generative models." In: *arXiv preprint arXiv:1709.01179*.

Chen, Jingbang, Yian Wang, Xingwei Qu, Shuangjia Zheng, Yaodong Yang, Hao Dong, and Jie Fu (2023). "Mixup-Augmented Meta-Learning for Sample-Efficient Fine-Tuning of Protein Simulators." In: *arXiv preprint arXiv:2308.15116*.

Chen, Liang-Chieh, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille (2014). "Semantic image segmentation with deep convolutional nets and fully connected crfs." In: *arXiv preprint arXiv:1412.7062*.

Chen, Tian Qi, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud (2018b). "Neural ordinary differential equations." In: *Advances in Neural Information Processing Systems*, pp. 6572–6583.

Chua, Kurtland, Roberto Calandra, Rowan McAllister, and Sergey Levine (2018). "Deep reinforcement learning in a handful of trials using probabilistic dynamics models." In: *arXiv preprint arXiv:1805.12114*.

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling." In: *arXiv preprint arXiv:1412.3555*.

Cohen, Taco S. and Max Welling (2017). "Steerable CNNs." In: *5th International Conference on Learning Representations, ICLR*.

Cohen, Taco and Max Welling (2016). "Group Equivariant Convolutional Networks." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger.

Coskun, Huseyin, Felix Achilles, Robert DiPietro, Nassir Navab, and Federico Tombari (2017). "Long short-term memory kalman filters: Recurrent neural estimators for pose regularization." In: *arXiv preprint arXiv:1708.01885*.

Crick, Christopher and Avi Pfeffer (2002). "Loopy belief propagation as a basis for communication in sensor networks." In: *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., pp. 159–166.

Cui, Taoyong, Chenyu Tang, Mao Su, Shufei Zhang, Yuqiang Li, Lei Bai, Yuhan Dong, Xingao Gong, and Wanli Ouyang (2023). "GPIP: Geometry-enhanced Pre-training on Interatomic Potentials." In: *arXiv preprint arXiv:2309.15718*.

Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function." In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

Dayan, Peter, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel (1995). "The helmholtz machine." In: *Neural computation* 7.5, pp. 889–904.

De Cao, Nicola and Thomas Kipf (2018). "MolGAN: An implicit generative model for small molecular graphs." In: *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*.

Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst (2016). "Convolutional neural networks on graphs with fast localized spectral filtering." In: *Advances in Neural Information Processing Systems*, pp. 3844–3852.

Deng, Zhiwei, Arash Vahdat, Hexiang Hu, and Greg Mori (2016). "Structure inference machines: Recurrent neural networks for analyzing relations in group activity recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4772–4781.

Dey, Neel, Antong Chen, and Soheil Ghafurian (2021). "Group Equivariant Generative Adversarial Networks." In: *International Conference on Learning Representations*.

Dhakal, Ashwin, Rajan Gyawali, and Jianlin Cheng (2023). "Predicting Protein-Ligand Binding Structure Using E (n) Equivariant Graph Neural Networks." In: *bioRxiv*, pp. 2023–08.

Dieleman, Sander, Jeffrey De Fauw, and Koray Kavukcuoglu (2016). "Exploiting Cyclic Symmetry in Convolutional Neural Networks." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR.org, pp. 1889–1898.

Dinh, Laurent, David Krueger, and Yoshua Bengio (2015). "NICE: Non-linear independent components estimation." In: *3rd International Conference on Learning Representations, ICLR, Workshop Track Proceedings*.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2017). "Density estimation using Real NVP." In: *5th International Conference on Learning Representations, ICLR*.

Falkovich, Gregory (2011). *Fluid mechanics: A short course for physicists*. Cambridge University Press.

Finlay, Chris, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M. Oberman (2020). "How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization." In: *Proceedings of the 37th International Conference on Machine Learning, ICML*.

Finzi, Marc, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson (2020a). "Generalizing Convolutional Neural Networks for Equivariance to Lie Groups on Arbitrary Continuous Data." In: *Proceedings of the 37th International Conference on Machine Learning, ICML*.

— (2020b). "Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data." In: *arXiv preprint arXiv:2002.12880*.

Fuchs, Fabian, Daniel Worrall, Volker Fischer, and Max Welling (2020). "Se (3)-transformers: 3d roto-translation equivariant attention networks." In: *Advances in neural information processing systems* 33, pp. 1970–1981.

Gallager, Robert (1962). "Low-density parity-check codes." In: *IRE Transactions on information theory* 8.1, pp. 21–28.

Ganea, Octavian-Eugen, Lagnajit Pattanaik, Connor W Coley, Regina Barzilay, Klavs F Jensen, William H Green, and Tommi S Jaakkola (2021). "GeoMol: Torsional Geometric Generation of Molecular 3D Conformer Ensembles." In: *arXiv preprint arXiv:2106.07802*.

Gao, Xiang, Weihao Gao, Wenzhi Xiao, Zhirui Wang, Chong Wang, and Liang Xiang (2022). "Supervised Pretraining for Molecular Force Fields and Properties Prediction." In: *arXiv preprint arXiv:2211.14429*.

Garcia Satorras, Victor, Zeynep Akata, and Max Welling (2019). "Combining generative and discriminative models for hybrid inference." In: *Advances in Neural Information Processing Systems* 32.

Gebauer, Niklas W. A., Michael Gastegger, and Kristof Schütt (2019). "Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*.

Gebauer, Niklas WA, Michael Gastegger, Stefaan SP Hessmann, Klaus-Robert Müller, and Kristof T Schütt (2021). "Inverse design of 3d molecular structures with conditional generative neural networks." In: *arXiv preprint arXiv:2109.04824*.

Gebauer, Niklas WA, Michael Gastegger, and Kristof T Schütt (2018). "Generating equilibrium molecules with deep neural networks." In: *arXiv preprint arXiv:1810.11347*.

Gilbert, Edgar N (1960). "Capacity of a burst-noise channel." In: *Bell system technical journal* 39.5, pp. 1253–1265.

Gilmer, Justin, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl (2017). "Neural message passing for quantum chemistry." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 1263–1272.

Gómez-Bombarelli, Rafael, David Duvenaud, José Miguel Hernández-Lobato, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik (2016). "Automatic chemical design using a data-driven continuous representation of molecules." In: *CoRR* abs/1610.02415.

Govindarajan, Prashant, Santiago Miret, Jarrid Rector-Brooks, Mariano Phielipp, Janarthanan Rajendran, and Sarath Chandar (2023). "Behavioral Cloning for Crystal Design." In: *Workshop on"Machine Learning for Materials"ICLR 2023*.

Grathwohl, Will, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud (2018). "Ffjord: Free-form continuous dynamics for scalable reversible generative models." In: *arXiv preprint arXiv:1810.01367*.

Grover, Aditya, Aaron Zweig, and Stefano Ermon (2019). "Graphite: Iterative generative modeling of graphs." In: *International conference on machine learning*. PMLR, pp. 2434–2444.

Grzeszczuk, Radek, Demetri Terzopoulos, and Geoffrey Hinton (1998). "Neuroanimator: Fast neural network emulation and control of physics-based models." In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 9–20.

Guan, Jiaqi, Wesley Wei Qian, Wei-Ying Ma, Jianzhu Ma, and Jian Peng (2021). "Energy-inspired molecular conformation optimization." In: *international conference on learning representations*.

Guan, Jiaqi, Wesley Wei Qian, Xingang Peng, Yufeng Su, Jian Peng, and Jianzhu Ma (2023). "3d equivariant diffusion for target-aware molecule generation and affinity prediction." In: *arXiv preprint arXiv:2303.03543*.

Haarnoja, Tuomas, Anurag Ajay, Sergey Levine, and Pieter Abbeel (2016). "Backprop kf: Learning discriminative deterministic state estimators." In: *Advances in Neural Information Processing Systems*, pp. 4376–4384.

Henaff, Mikael, Joan Bruna, and Yann LeCun (2015). "Deep convolutional networks on graph-structured data." In: *arXiv preprint arXiv:1506.05163*.

Ho, Jonathan, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel (2019). "Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design." In: *36th International Conference on Machine Learning*.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). "Denoising Diffusion Probabilistic Models." In: *CoRR* abs/2006.11239.

Hoffmann, Moritz and Frank Noé (2019). "Generating valid Euclidean distance matrices." In: *arXiv preprint arXiv:1910.03131*.

Hoogeboom, Emiel, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling (2021). "Argmax Flows and Multinomial Diffusion: Towards Non-Autoregressive Language Models." In: *CoRR* abs/2102.05379.

Hoogeboom, Emiel, Vıctor Garcia Satorras, Clément Vignac, and Max Welling (2022). "Equivariant diffusion for molecule generation in 3d." In: *International conference on machine learning*. PMLR, pp. 8867–8887.

Horie, Masanobu, Naoki Morita, Toshiaki Hishinuma, Yu Ihara, and Naoto Mitsume (2020). "Isometric transformation invariant and equivariant graph convolutional networks." In: *arXiv preprint arXiv:2005.06316*.

Hutchinson, Michael, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, and Hyunjik Kim (2020). "LieTransformer: Equivariant self-attention for Lie Groups." In: *CoRR* abs/2012.10885.

Igashov, Ilia, Hannes Stärk, Clément Vignac, Victor Garcia Satorras, Pascal Frossard, Max Welling, Michael Bronstein, and Bruno Correia (2022). "Equivariant 3d-conditional diffusion models for molecular linker design." In: *arXiv preprint arXiv:2210.05274*.

Isert, Clemens, Kenneth Atz, and Gisbert Schneider (2023). "Structure-based drug design with geometric deep learning." In: *Current Opinion in Structural Biology* 79, p. 102548.

Johnson, Matthew, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta (2016). "Composing graphical models with neural networks for structured representations and fast inference." In: *Advances in neural information processing systems*, pp. 2946–2954.

Kalman, Rudolph Emil (1960). "A new approach to linear filtering and prediction problems." In: *Journal of basic Engineering* 82.1, pp. 35–45.

Kim, Hyeji, Yihan Jiang, Ranvir Rana, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath (2018). "Communication algorithms via deep learning." In: *arXiv preprint arXiv:1805.09317*.

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

Kingma, Diederik P, Tim Salimans, Ben Poole, and Jonathan Ho (2021). "Variational diffusion models." In: *arXiv preprint arXiv:2107.00630* 2.

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114*.

Kingma, Durk P and Prafulla Dhariwal (2018). "Glow: Generative flow with invertible 1x1 convolutions." In: *Advances in Neural Information Processing Systems*, pp. 10236–10245.

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

— (2016b). "Variational graph auto-encoders." In: *arXiv preprint arXiv:1611.07308*.

Kipf, Thomas, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel (2018). "Neural relational inference for interacting systems." In: *arXiv preprint arXiv:1802.04687*.

Klambauer, Günter, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter (2017). "Self-normalizing neural networks." In: *Advances in neural information processing systems*, pp. 971–980.

Klicpera, Johannes, Shankari Giri, Johannes T Margraf, and Stephan Günnemann (2020a). "Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules." In: *arXiv preprint arXiv:2011.14115*.

Klicpera, Johannes, Janek Groß, and Stephan Günnemann (2020b). "Directional Message Passing for Molecular Graphs." In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

— (2020c). "Directional Message Passing for Molecular Graphs." In: *8th International Conference on Learning Representations, ICLR*.

Klicpera, Johannes, Janek Groß, and Stephan Günnemann (2020d). "Directional message passing for molecular graphs." In: *arXiv preprint arXiv:2003.03123*.

Kobyzev, Ivan, Simon JD Prince, and Marcus A Brubaker (2020). "Normalizing flows: An introduction and review of current methods." In: *IEEE transactions on pattern analysis and machine intelligence* 43.11, pp. 3964–3979.

Köhler, Jonas, Leon Klein, and Frank Noé (2019). "Equivariant Flows: sampling configurations for multi-body systems with symmetric energies." In: *CoRR* abs/1910.00753.

— (2020a). "Equivariant Flows: Exact Likelihood Generative Learning for Symmetric Densities." In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 5361–5370.

— (2020b). "Equivariant Flows: Exact Likelihood Generative Learning for Symmetric Densities." In: *Proceedings of the 37th International Conference on Machine Learning, ICML*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 5361–5370.

Koller, Daphne, Nir Friedman, and Francis Bach (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Kondor, Risi, Hy Truong Son, Horace Pan, Brandon Anderson, and Shubhendu Trivedi (2018). "Covariant compositional networks for learning graphs." In: *arXiv preprint arXiv:1801.02144*.

Kosiorek, Adam R, Hyunjik Kim, and Danilo J Rezende (2020). "Conditional set generation with transformers." In: *Workshop on Object-Oriented Learning at ICML 2020*.

Krawczuk, Igor, Pedro Abranches, Andreas Loukas, and Volkan Cevher (2021). *GG-GAN: A Geometric Graph Generative Adversarial Network*. URL: https://openreview.net/forum?id=qiAxL3Xqx1o.

Kuck, Jonathan, Shuvam Chakraborty, Hao Tang, Rachel Luo, Jiaming Song, Ashish Sabharwal, and Stefano Ermon (2020). "Belief Propagation Neural Networks." In: *arXiv preprint arXiv:2007.00295*.

Labbe, Roger (2014). *Kalman and bayesian filters in python*.

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

Li, Shuo-Hui, Chen-Xiao Dong, Linfeng Zhang, and Lei Wang (2019). "Neural Canonical Transformation with Symplectic Flows." In: *CoRR* abs/1910.00024. arXiv: 1910.00024. URL: http://arxiv.org/abs/1910.00024.

Li, Yujia, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel (2015). "Gated graph sequence neural networks." In: *arXiv preprint arXiv:1511.05493*.

Liao, Renjie, Yujia Li, Yang Song, Shenlong Wang, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel (2019). "Efficient Graph Generation with Graph Recurrent Attention Networks." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*.

Limoyo, Oliver, Filip Marić, Matthew Giamou, Petra Alexson, Ivan Petrović, and Jonathan Kelly (2023). "Euclidean Equivariant Models for Generative Graphical Inverse Kinematics." In: *arXiv preprint arXiv:2307.01902*.

Lin, Jianhua (1991). "Divergence measures based on the Shannon entropy." In: *IEEE Transactions on Information theory* 37.1, pp. 145–151.

Liu, Jenny, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky (2019). "Graph normalizing flows." In: *Advances in Neural Information Processing Systems*, pp. 13578–13588.

Liu, Qi, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L Gaunt (2018). "Constrained graph variational autoencoders for molecule design." In: *arXiv preprint arXiv:1805.09076*.

Liu, Ye-Hua and David Poulin (2019). "Neural belief-propagation decoders for quantum error-correcting codes." In: *Physical review letters* 122.20, p. 200501.

Ljung, Lennart (1979). "Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems." In: *IEEE Transactions on Automatic Control* 24.1, pp. 36–50.

Loeliger, H-A (2004). "An introduction to factor graphs." In: *IEEE Signal Processing Magazine* 21.1, pp. 28–41.

Luo, Shitong, Chence Shi, Minkai Xu, and Jian Tang (2021). "Predicting Molecular Conformation via Dynamic Graph Score Matching." In: *Advances in Neural Information Processing Systems* 34.

MacKay, David JC (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

MacKay, David JC and Radford M Neal (1996). "Near Shannon limit performance of low density parity check codes." In: *Electronics letters* 32.18, pp. 1645–1646.

MacKay, David and Error-Correcting Codes (2009). "David MacKay's Gallager code resources." In: *URL: http://www. inference. phy. cam. ac. uk/mackay/CodesFiles. html*.

Mahmud, Sajid, Alex Morehead, and Jianlin Cheng (2023). "Accurate prediction of protein tertiary structural changes induced by single-site mutations with equivariant graph neural networks." In: *bioRxiv*, pp. 2023–10.

Marino, Joseph, Yisong Yue, and Stephan Mandt (2018). "Iterative amortized inference." In: *arXiv preprint arXiv:1807.09356*.

Masters, Matthew, Amr H Mahmoud, Yao Wei, and Markus Alexander Lill (2022). "Deep learning model for flexible and efficient protein-ligand docking." In: *ICLR2022 Machine Learning for Drug Discovery*.

McEliece, Robert J., David J. C. MacKay, and Jung-Fu Cheng (1998). "Turbo decoding as an instance of Pearl's" belief propagation" algorithm." In: *IEEE Journal on selected areas in communications* 16.2, pp. 140–152.

Miller, Benjamin Kurt, Mario Geiger, Tess E Smidt, and Frank Noé (2020). "Relevance of rotationally equivariant convolutions for predicting molecular properties." In: *arXiv preprint arXiv:2008.08461*.

Mirowski, Piotr and Yann LeCun (2009). "Dynamic factor graphs for time series modeling." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 128–143.

Mitton, Joshua, Hans M Senn, Klaas Wynne, and Roderick Murray-Smith (2021). "A Graph VAE and Graph Transformer Approach to Generating Molecular Graphs." In: *arXiv preprint arXiv:2104.04345*.

Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*. MIT press.

Murphy, Kevin, Yair Weiss, and Michael I Jordan (2013). "Loopy belief propagation for approximate inference: An empirical study." In: *arXiv preprint arXiv:1301.6725*.

Nachmani, Eliya, Yair Be'ery, and David Burshtein (2016). "Learning to decode linear codes using deep learning." In: *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, pp. 341–346.

Nagabandi, Anusha, Gregory Kahn, Ronald S Fearing, and Sergey Levine (2018). "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 7559–7566.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines." In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Nauata, Nelson, Hexiang Hu, Guang-Tong Zhou, Zhiwei Deng, Zicheng Liao, and Greg Mori (2018). "Structured Label Inference for Visual Understanding." In: *arXiv preprint arXiv:1802.06459*.

Neal, Radford M et al. (2011). "MCMC using Hamiltonian dynamics." In: *Handbook of Markov Chain Monte Carlo* 2.11, p. 2.

Nichol, Alex and Prafulla Dhariwal (2021). "Improved denoising diffusion probabilistic models." In: *arXiv preprint arXiv:2102.09672*.

Noé, Frank, Simon Olsson, Jonas Köhler, and Hao Wu (2019). "Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning." In: *Science* 365.6457.

Noé, Frank, Simon Olsson, and Jonas Köhler Hao Wu (2018). "Boltzmann Generators - Sampling Equilibrium States of Many-Body Systems with Deep Learn-

ing." In: *CoRR* abs/1812.01729. arXiv: 1812.01729. URL: http://arxiv.org/abs/1812.01729.

Nwankpa, Chigozie, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall (2018). "Activation functions: Comparison of trends in practice and research for deep learning." In: *arXiv preprint arXiv:1811.03378*.

Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu (2016). "Wavenet: A generative model for raw audio." In: *arXiv preprint arXiv:1609.03499*.

Pearl, Judea (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann.

— (2014). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier.

Pratik, Kumar, Bhaskar D Rao, and Max Welling (2020). "RE-MIMO: Recurrent and permutation equivariant neural MIMO detection." In: *IEEE Transactions on Signal Processing* 69, pp. 459–473.

Putzky, Patrick and Max Welling (2017). "Recurrent inference machines for solving inverse problems." In: *arXiv preprint arXiv:1706.04008*.

Ramachandran, Prajit, Barret Zoph, and Quoc V Le (2017). "Searching for activation functions." In: *arXiv preprint arXiv:1710.05941*.

Ramakrishnan, Raghunathan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld (2014). "Quantum chemistry structures and properties of 134 kilo molecules." In: *Scientific data* 1.1, pp. 1–7.

Ramesh, Aditya, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever (2021). "Zero-shot text-to-image generation." In: *International Conference on Machine Learning*. PMLR, pp. 8821–8831.

Rauch, Herbert E, CT Striebel, and F Tung (1965). "Maximum likelihood estimates of linear dynamic systems." In: *AIAA journal* 3.8, pp. 1445–1450.

Revach, Guy, Nir Shlezinger, Xiaoyong Ni, Adria Lopez Escoriza, Ruud JG Van Sloun, and Yonina C Eldar (2022). "KalmanNet: Neural network aided Kalman filtering for partially known dynamics." In: *IEEE Transactions on Signal Processing* 70, pp. 1532–1547.

Rezende, Danilo Jimenez and Shakir Mohamed (2015a). "Variational inference with normalizing flows." In: *arXiv preprint arXiv:1505.05770*.

Rezende, Danilo Jimenez, Sébastien Racanière, Irina Higgins, and Peter Toth (2019). "Equivariant Hamiltonian Flows." In: *CoRR* abs/1909.13739.

Rezende, Danilo and Shakir Mohamed (2015b). "Variational Inference with Normalizing Flows." In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. Proceedings of Machine Learning Research. PMLR, pp. 1530–1538.

Richardson, Matthew and Pedro Domingos (2006). "Markov logic networks." In: *Machine learning* 62.1-2, pp. 107–136.

Rippel, Oren and Ryan Prescott Adams (2013). "High-dimensional probability estimation with deep density models." In: *arXiv preprint arXiv:1302.5125*.

Romero, David W. and Jean-Baptiste Cordonnier (2021). "Group Equivariant Stand-Alone Self-Attention For Vision." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=JkfYjnOEo6M.

Rozenberg, Eyal and Daniel Freedman (2022). "Molecule Generation for Target Receptor Binding via Continuous Normalizing Flows." In.

Salimans, Tim, Diederik Kingma, and Max Welling (2015). "Markov chain monte carlo and variational inference: Bridging the gap." In: *International Conference on Machine Learning*, pp. 1218–1226.

Satorras, Victor Garcia, Zeynep Akata, and Max Welling (2019). "Combining Generative and Discriminative Models for Hybrid Inference." In: *arXiv preprint arXiv:1906.02547*.

Satorras, Victor Garcia, Emiel Hoogeboom, Fabian Fuchs, Ingmar Posner, and Max Welling (2021a). "E(n) Equivariant Normalizing Flows." In: *Advances in Neural Information Processing Systems* 34.

Satorras, Victor Garcia, Emiel Hoogeboom, and Max Welling (2021b). "E (n) equivariant graph neural networks." In: *International conference on machine learning*. PMLR, pp. 9323–9332.

Satorras, Victor Garcia and Max Welling (2021). "Neural enhanced belief propagation on factor graphs." In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 685–693.

Schmidhuber, Jürgen (1987). "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook." PhD thesis. Technische Universität München.

Schneuing, Arne, Yuanqi Du, Charles Harris, Arian Jamasb, Ilia Igashov, Weitao Du, Tom Blundell, Pietro Lió, Carla Gomes, Max Welling, et al. (2022). "Structure-based drug design with equivariant diffusion models." In: *arXiv preprint arXiv:2210.13695*.

Schütt, Kristof T, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko (2017a). "Quantum-chemical insights from deep tensor neural networks." In: *Nature communications* 8.1, pp. 1–8.

Schütt, Kristof T, Pieter-Jan Kindermans, Huziel E Sauceda, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller (2017b). "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions." In: *arXiv preprint arXiv:1706.08566*.

Schütt, Kristof, Oliver Unke, and Michael Gastegger (2021). "Equivariant message passing for the prediction of tensorial properties and molecular spectra." In: *International Conference on Machine Learning*. PMLR, pp. 9377–9388.

Serviansky, Hadar, Nimrod Segol, Jonathan Shlomi, Kyle Cranmer, Eilam Gross, Haggai Maron, and Yaron Lipman (2020). "Set2graph: Learning graphs from sets." In: *Advances in Neural Information Processing Systems* 33.

Sestak, Florian, Lisa Schneckenreiter, Sepp Hochreiter, Andreas Mayr, and Günter Klambauer (2023). "VN-EGNN: Equivariant Graph Neural Networks with Virtual Nodes Enhance Protein Binding Site Identification." In: *NeurIPS 2023 Workshop on New Frontiers of AI for Drug Discovery and Development*.

Shi, Chence, Shitong Luo, Minkai Xu, and Jian Tang (2021). "Learning Gradient Fields for Molecular Conformation Generation." In: *Proceedings of the 38th International Conference on Machine Learning, ICML*. Ed. by Marina Meila and Tong Zhang.

Shlezinger, Nir, Jay Whang, Yonina C Eldar, and Alexandros G Dimakis (2023). "Model-based deep learning." In: *Proceedings of the IEEE*.

Simm, Gregor N. C., Robert Pinsler, Gábor Csányi, and José Miguel Hernández-Lobato (2021). "Symmetry-Aware Actor-Critic for 3D Molecular Design." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=jEYKjPE1xYN.

Simm, Gregor NC and José Miguel Hernández-Lobato (2019). "A generative model for molecular distance geometry." In: *arXiv preprint arXiv:1909.11459*.

Simonovsky, Martin and Nikos Komodakis (2018). "Graphvae: Towards generation of small graphs using variational autoencoders." In: *International Conference on Artificial Neural Networks*. Springer, pp. 412–422.

Sohl-Dickstein, Jascha, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli (2015). "Deep Unsupervised Learning using Nonequilibrium Thermodynamics." In: *Proceedings of the 32nd International Conference on Machine Learning, ICML*. Ed. by Francis R. Bach and David M. Blei.

Song, Yang and Stefano Ermon (2019). "Generative Modeling by Estimating Gradients of the Data Distribution." In: *CoRR* abs/1907.05600. arXiv: 1907.05600. URL: http://arxiv.org/abs/1907.05600.

Song, Zhenqiao, Yunlong Zhao, Yufei Song, Wenxian Shi, Yang Yang, and Lei Li (2023). "Joint Design of Protein Sequence and Structure based on Motifs." In: *arXiv preprint arXiv:2310.02546*.

Srinivasan, Balasubramaniam and Bruno Ribeiro (2019). "On the equivalence between positional node embeddings and structural graph representations." In: *arXiv preprint arXiv:1910.00452*.

Stärk, Hannes, Octavian Ganea, Lagnajit Pattanaik, Regina Barzilay, and Tommi Jaakkola (2022). "Equibind: Geometric deep learning for drug binding structure prediction." In: *International conference on machine learning*. PMLR, pp. 20503–20521.

Thölke, Philipp and Gianni De Fabritiis (2022). "Torchmd-net: equivariant transformers for neural network based molecular potentials." In: *arXiv preprint arXiv:2202.02541*.

Thomas, Nathaniel, Tess Smidt, Steven M. Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley (2018a). "Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds." In: *CoRR* abs/1802.08219.

Thomas, Nathaniel, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley (2018b). "Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds." In: *arXiv preprint arXiv:1802.08219*.

Trippe, Brian L, Jason Yim, Doug Tischer, David Baker, Tamara Broderick, Regina Barzilay, and Tommi Jaakkola (2022). "Diffusion probabilistic modeling of

protein backbones in 3d for the motif-scaffolding problem." In: *arXiv preprint arXiv:2206.04119*.

Uy, Mikaela Angelina, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung (2019). "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data." In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1588–1597.

Vignac, Clement and Pascal Frossard (2021). "Top-N: Equivariant set and graph generation without exchangeability." In: *arXiv preprint arXiv:2110.02096*.

Wan, Eric A and Rudolph Van Der Merwe (2000). "The unscented Kalman filter for nonlinear estimation." In: *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. Ieee, pp. 153–158.

Watters, Nicholas, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti (2017). "Visual interaction networks: Learning a physics simulator from video." In: *Advances in neural information processing systems* 30, pp. 4539–4547.

Wei, Shih-En, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh (2016). "Convolutional pose machines." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4724–4732.

Weiler, Maurice and Gabriele Cesa (2019). "General E(2)-Equivariant Steerable CNNs." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett.

Welch, Greg, Gary Bishop, et al. (1995). "An introduction to the Kalman filter." In: *Chapel Hill, NC, USA*.

Wu, Jiaxiang, Tao Shen, Haidong Lan, Yatao Bian, and Junzhou Huang (2021). "SE (3)-equivariant energy-based models for end-to-end protein folding." In: *bioRxiv*, pp. 2021–06.

Xu, Minkai, Shitong Luo, Yoshua Bengio, Jian Peng, and Jian Tang (2021a). "Learning Neural Generative Dynamics for Molecular Conformation Generation." In: *CoRR* abs/2102.10240.

Xu, Minkai, Wujie Wang, Shitong Luo, Chence Shi, Yoshua Bengio, Rafael Gomez-Bombarelli, and Jian Tang (2021b). "An End-to-End Framework for Molecular Conformation Generation via Bilevel Programming." In: *arXiv preprint arXiv:2105.07246*.

Xu, Minkai, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang (2022). "GeoDiff: A Geometric Diffusion Model for Molecular Conformation Generation." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=PzcvxEMzvQC.

Xu, Yilun, Yang Song, Sahaj Garg, Linyuan Gong, Rui Shu, Aditya Grover, and Stefano Ermon (2021c). "Anytime Sampling for Autoregressive Models via Ordered Autoencoding." In: *9th International Conference on Learning Representations, ICLR*.

Yedidia, Jonathan S, William T Freeman, and Yair Weiss (2003). "Understanding belief propagation and its generalizations." In: *Exploring artificial intelligence in the new millennium* 8, pp. 236–239.

Yoon, KiJung, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow (2018). "Inference in Probabilistic Graphical Models by Graph Neural Networks." In: *arXiv preprint arXiv:1803.07710*.

You, Jiaxuan, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec (2018). "Graphrnn: Generating realistic graphs with deep auto-regressive models." In: *arXiv preprint arXiv:1802.08773*.

Zeni, Claudio, Robert Pinsler, Daniel Zügner, Andrew Fowler, Matthew Horton, Xiang Fu, Sasha Shysheya, Jonathan Crabbé, Lixin Sun, Jake Smith, et al. (2023). "MatterGen: a generative model for inorganic materials design." In: *arXiv preprint arXiv:2312.03687*.

Zhang, Yang, Wenbing Huang, Zhewei Wei, Ye Yuan, and Zhaohan Ding (2023). "EquiPocket: an E (3)-Equivariant Geometric Graph Neural Network for Ligand Binding Site Prediction." In: *arXiv preprint arXiv:2302.12177*.

Zhang, Zhen, Fan Wu, and Wee Sun Lee (2019). "Factor Graph Neural Network." In: *arXiv preprint arXiv:1906.00554*.

Zhang, Zuobai, Minghao Xu, Arian Jamasb, Vijil Chenthamarakshan, Aurelie Lozano, Payel Das, and Jian Tang (2022). "Protein representation learning by geometric structure pretraining." In: *arXiv preprint arXiv:2203.06125*.

Zheng, Shuai, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr (2015). "Conditional random fields as recurrent neural networks." In: *Proceedings of the IEEE international conference on computer vision*, pp. 1529–1537.

Part IV

APPENDICES

A

HYBRID INFERENCE IN MARKOV CHAINS

A.1 EQUATION DETAILS THE FOR LINEAR DYNAMICS EXPERIMENT

Linear and Gaussian dataset matrices:

The differential equations that describe the dynamics are ($c = 0.06$, $\tau = 0.17$):

$$\frac{\partial \mathbf{p}}{\partial t} = \mathbf{v}, \qquad \frac{\partial \mathbf{v}}{\partial t} = \mathbf{a} - c\mathbf{v}, \qquad \frac{\partial \mathbf{a}}{\partial t} = -\tau\mathbf{v} \tag{71}$$

Therefore, the dynamics matrix is defined as:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -c & 1 \\ 0 & -\tau & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{a} \end{bmatrix} \tag{72}$$

Using the following Taylor expansion we find the transition matrix $\tilde{\mathbf{F}}$

$$\tilde{\mathbf{F}} = \mathbf{I} + \sum_{k=1}^{K} \frac{(\mathbf{A}\Delta t)^k}{k!} \tag{73}$$

The transition matrix $\tilde{\mathbf{F}}$ for each dimension is:

$$\tilde{\mathbf{F}} = \begin{bmatrix} 1 & \Delta t - \frac{c}{2}\Delta t^2 & \frac{\Delta t^2}{2} \\ 0 & 1 - c\Delta t + \frac{c^2 - \tau}{2}\Delta t^2 & \Delta t - \frac{c}{2}\Delta t^2 \\ 0 & -\tau\Delta t + \frac{\tau c}{2}\Delta t^2 & 1 - \frac{\tau}{2}\Delta t^2 \end{bmatrix} \tag{74}$$

Then, the transition matrix and noise distributions are:

$$\mathbf{F} = \begin{bmatrix} \tilde{\mathbf{F}} & \mathbf{o} \\ \mathbf{o} & \tilde{\mathbf{F}} \end{bmatrix}, \tilde{\mathbf{Q}} = \begin{bmatrix} \Delta t/3 & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & 3\Delta t \end{bmatrix}, \mathbf{Q} = 0.1^2 \begin{bmatrix} \tilde{\mathbf{Q}} & 0 \\ 0 & \tilde{\mathbf{Q}} \end{bmatrix},$$

The measurement matrix and noise distribution are:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \mathbf{R} = 0.5^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Linear and Gaussian matrices used for hybrid and GM-messages models:

The differential equations that describe the dynamics are:

$$\frac{\partial \mathbf{p}}{\partial t} = \mathbf{v}, \qquad \frac{\partial \mathbf{v}}{\partial t} = 0, \qquad \frac{\partial \mathbf{a}}{\partial t} = 0 \tag{75}$$

Then the transition matrix given the last equations is:

$$\tilde{\mathbf{F}} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{76}$$

And the transition matrix and noise distributions are:

$$\mathbf{F} = \begin{bmatrix} \tilde{\mathbf{F}} & 0 \\ 0 & \tilde{\mathbf{F}} \end{bmatrix}, \tilde{\mathbf{Q}} = \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \mathbf{Q} = \begin{bmatrix} \sigma^2 \tilde{\mathbf{Q}} & 0 \\ 0 & \sigma^2 \tilde{\mathbf{Q}} \end{bmatrix}, \tag{77}$$

Finally the measurement distribution matrices are:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \mathbf{R} = 0.5^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Such that the only parameter to optimize from the graphical model is the variance of the transition noise distribution σ

A.1.1 *Equation details for the NCLT dataset*

For the NCLT dataset we use the uniform velocity motion equations. The differential equations that describe the dynamics are:

$$\frac{\partial \mathbf{p}}{\partial t} = v, \qquad \frac{\partial \mathbf{v}}{\partial t} = 0, \qquad \frac{\partial \mathbf{a}}{\partial t} = 0 \tag{78}$$

Such that the transition matrix for one component is:

$$\tilde{\mathbf{F}} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \tag{79}$$

And the transition matrix and noise distribution are:

$$\mathbf{F} = \begin{bmatrix} \tilde{\mathbf{F}} & 0 \\ 0 & \tilde{\mathbf{F}} \end{bmatrix}, \tilde{\mathbf{Q}} = \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \mathbf{Q} = \begin{bmatrix} \sigma^2 \tilde{\mathbf{Q}} & 0 \\ 0 & \sigma^2 \tilde{\mathbf{Q}} \end{bmatrix}, \tag{80}$$

Finally the measurement distribution matrices are:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \mathbf{R} = \lambda^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Such that the only parameters to optimize from the graphical model is the variance of the noise and measurement distributions σ and λ.

B

E(N) EQUIVARIANT GRAPH NEURAL NETWORKS

B.1 EQUIVARIANCE PROOF

In this section we prove that our model is translation equivariant on $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_M) \in \mathbb{R}^{M \times n}$ for any translation vector $\mathbf{g} \in \mathbb{R}^n$ and it is rotation and reflection equivariant on $\mathbf{X}$ for any orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$. Let $\mathbf{QX}$ be shorthand for $(\mathbf{Qx}_1, \ldots, \mathbf{Qx}_M)$ and $\mathbf{X} + \mathbf{g}$ be shorthand for $(\mathbf{x}_1 + \mathbf{g}, \ldots, \mathbf{x}_M + \mathbf{g})$. We will prove the model satisfies:

$$\mathbf{QX}^{l+1} + \mathbf{g}, \mathbf{H}^{l+1} = \text{EGCL}(\mathbf{QX}^l + \mathbf{g}, \mathbf{H}^l)$$

We will analyze how a translation and rotation of the input coordinates propagates through our model. We start assuming $\mathbf{H}^0$ is invariant to $E(n)$ transformations on $\mathbf{X}$, in other words, we do not encode any information about the absolute position or orientation of $\mathbf{X}^0$ into $\mathbf{H}^0$. Then, the output $\mathbf{m}_{ij}$ of Equation 42 will be invariant too since the distance between two particles is invariant to translations $\|\mathbf{x}_i^l + \mathbf{g} - [\mathbf{x}_j^l + \mathbf{g}]\|^2 = \|\mathbf{x}_i^l - \mathbf{x}_j^l\|^2$, and it is invariant to rotations and reflections $\|\mathbf{Qx}_i^l - \mathbf{Qx}_j^l\|^2 = (\mathbf{x}_i^l - \mathbf{x}_j^l)^\top \mathbf{Q}^\top \mathbf{Q}(\mathbf{x}_i^l - \mathbf{x}_j^l) = (\mathbf{x}_i^l - \mathbf{x}_j^l)^\top \mathbf{I}(\mathbf{x}_i^l - \mathbf{x}_j^l) = \|\mathbf{x}_i^l - \mathbf{x}_j^l\|^2$ such that the edge operation becomes invariant:

$$\mathbf{m}_{i,j} = \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{Qx}_i^l + \mathbf{g} - [\mathbf{Qx}_j^l + \mathbf{g}]\right\|^2, \mathbf{a}_{ij}\right) = \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2, \mathbf{a}_{ij}\right)$$

The second equation of our model (eq. 43) that updates the coordinates $\mathbf{x}_i$ is $E(n)$ equivariant. Following, we prove its equivariance by showing that an $E(n)$ trans-

formation of the input leads to the same transformation of the output. Notice $\mathbf{m}_{ij}$ is already invariant as proven above. We want to show:

$$\mathbf{Q}\mathbf{x}_i^{l+1} + \mathbf{g} = \mathbf{Q}\mathbf{x}_i^l + \mathbf{g} + C\sum_{j\neq i}\left(\mathbf{Q}\mathbf{x}_i^l + \mathbf{g} - [\mathbf{Q}\mathbf{x}_j^l + \mathbf{g}]\right)\phi_x\left(\mathbf{m}_{i,j}\right)$$

Derivation.

$$\begin{aligned}
&\mathbf{Q}\mathbf{x}_i^l + \mathbf{g} + C\sum_{j\neq i}\left(\mathbf{Q}\mathbf{x}_i^l + \mathbf{g} - \mathbf{Q}\mathbf{x}_j^l - \mathbf{g}\right)\phi_x\left(\mathbf{m}_{i,j}\right)\\
&= \mathbf{Q}\mathbf{x}_i^l + \mathbf{g} + \mathbf{Q}C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{i,j}\right)\\
&= \mathbf{Q}\left(\mathbf{x}_i^l + C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{i,j}\right)\right) + \mathbf{g}\\
&= \mathbf{Q}\mathbf{x}_i^{l+1} + \mathbf{g}
\end{aligned}$$

Therefore, we have proven that rotating and translating $\mathbf{x}_i^l$ results in the same rotation and translation on $\mathbf{x}_i^{l+1}$ at the output of Equation 43.

Furthermore equations 44 and 45 only depend on $\mathbf{m}_{ij}$ and $\mathbf{h}_i^l$ which as saw at the beginning of this proof, are $E(n)$ invariant, therefore the output of Equation 45 $\mathbf{h}_i^{l+1}$ will be invariant too. Thus concluding that a transformation $\mathbf{Q}\mathbf{x}_i^l + g$ on $\mathbf{x}_i^l$ will result in the same transformation on $\mathbf{x}_i^{l+1}$ while $\mathbf{h}_i^{l+1}$ will remain invariant to it such that $\mathbf{Q}\mathbf{X}^{l+1} + \mathbf{g}, \mathbf{H}^{l+1} = \text{EGCL}(\mathbf{Q}\mathbf{X}^l + \mathbf{g}, \mathbf{H}^l)$ is satisfied.

B.2 RE-FORMULATION FOR VELOCITY TYPE INPUTS

In this section we write down the EGNN transformation layer $\mathbf{H}^{l+1}, \mathbf{X}^{l+1}, \mathbf{V}^{l+1} = \text{EGCL}[\mathbf{H}^l, \mathbf{X}^l, \mathbf{V}^{\text{init}}, \mathcal{E}]$ that can take in velocity input and output channels. We also prove it remains $\text{E}(n)$ equivariant.

$$
\begin{aligned}
\mathbf{m}_{ij} &= \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2, \mathbf{a}_{ij}\right) \\
\mathbf{v}_i^{l+1} &= \phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^{\text{init}} + C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \\
\mathbf{x}_i^{l+1} &= \mathbf{x}_i^l + \mathbf{v}_i^{l+1} \\
\mathbf{m}_i &= \sum_{j\neq i}\mathbf{m}_{ij} \\
\mathbf{h}_i^{l+1} &= \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right)
\end{aligned}
$$

B.2.1 *Equivariance proof for velocity type inputs*

In this subsection we prove that the velocity types input formulation of our model is also $\text{E}(n)$ equivariant on $\mathbf{X}$. More formally, for any translation vector $\mathbf{g} \in \mathbb{R}^n$ and for any orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$, the model should satisfy:

$$\mathbf{H}^{l+1}, \mathbf{Q}\mathbf{X}^{l+1} + \mathbf{g}, \mathbf{Q}\mathbf{V}^{l+1} = \text{EGCL}[\mathbf{H}^l, \mathbf{Q}\mathbf{X}^l + \mathbf{g}, \mathbf{Q}\mathbf{V}^{\text{init}}, \mathcal{E}]$$

In Appendix B.1 we already proved the equivariance of our EGNN (Section 5.3) when not including vector type inputs. In its velocity type inputs variant we only replaced its coordinate updates (Eq. 43) by Equation 46 that includes velocity. Since this is the only modification we will only prove that Equation 46 re-written below is equivariant.

$$
\begin{aligned}
\mathbf{v}_i^{l+1} &= \phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^{\text{init}} + C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \\
\mathbf{x}_i^{l+1} &= \mathbf{x}_i^l + \mathbf{v}_i^{l+1}
\end{aligned}
$$

First, we prove the first line preserves equivariance, that is we want to show:

$$\mathbf{Q}\mathbf{v}_i^{l+1} = \phi_v\left(\mathbf{h}_i^l\right)\mathbf{Q}\mathbf{v}_i^{\text{init}} + C\sum_{j\neq i}\left(\mathbf{Q}\mathbf{x}_i^l + \mathbf{g} - [\mathbf{Q}\mathbf{x}_j^l + \mathbf{g}]\right)\phi_x\left(\mathbf{m}_{ij}\right)$$

Derivation.

$$\phi_v\left(\mathbf{h}_i^l\right)\mathbf{Q}\mathbf{v}_i^{\text{init}} + C\sum_{j\neq i}\left(\mathbf{Q}\mathbf{x}_i^l + \mathbf{g} - [\mathbf{Q}\mathbf{x}_j^l + \mathbf{g}]\right)\phi_x\left(\mathbf{m}_{ij}\right) \tag{81}$$

$$= \mathbf{Q}\phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^{\text{init}} + \mathbf{Q}C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \tag{82}$$

$$= \mathbf{Q}\left(\phi_v\left(\mathbf{h}_i^l\right)\mathbf{v}_i^{\text{init}} + C\sum_{j\neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right)\right) \tag{83}$$

$$= \mathbf{Q}\mathbf{v}_i^{l+1} \tag{84}$$

Finally, it is straightforward to show the second equation is also equivariant, that is we want to show $\mathbf{Q}\mathbf{x}_i^{l+1} + \mathbf{g} = \mathbf{Q}\mathbf{x}_i^l + \mathbf{g} + \mathbf{Q}\mathbf{v}_i^{l+1}$

Derivation.

$$\begin{aligned}\mathbf{Q}\mathbf{x}_i^l + \mathbf{g} + \mathbf{Q}\mathbf{v}_i^{l+1} &= \mathbf{Q}\left(\mathbf{x}_i^l + \mathbf{v}_i^{l+1}\right) + \mathbf{g} \\ &= \mathbf{Q}\mathbf{x}_i^{l+1} + \mathbf{g}\end{aligned}$$

Concluding we showed that an $E(n)$ transformation on the input set of points results in the same transformation on the output set of points such that $\mathbf{H}^{l+1}, \mathbf{Q}\mathbf{X}^{l+1} + \mathbf{g}, \mathbf{Q}\mathbf{V}^{l+1} = \text{EGCL}[\mathbf{H}^l, \mathbf{Q}\mathbf{X}^l + \mathbf{g}, \mathbf{Q}\mathbf{V}^{\text{init}}, \mathcal{E}]$ is satisfied.

B.3 IMPLEMENTATION DETAILS

In this Appendix section we describe the implementation details of the experiments. First, we describe those parts of our model that are the same across all experiments. Our EGNN model from Section 5.3 contains the following three main learnable functions.

The edge function ϕ_e (Eq. 42) is a two layers MLP with two Swish non-linearities: Input $\rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() $\rightarrow$ Swish() } $\rightarrow$ Output.

The coordinate function ϕ_x (Eq. 43) consists of a two layers MLP with one non-linearity: $\mathbf{m}_{ij} \rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() } $\rightarrow$ Output.

The node function ϕ_h (Eq. 45) consists of a two layers MLP with one non-linearity and a residual connection: $[\mathbf{h}_i^l, \mathbf{m}_i] \rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() $\rightarrow$ Addition($\mathbf{h}_i^l$) } $\rightarrow \mathbf{h}_i^{l+1}$.

These functions are used in our EGNN across all experiments. Notice the GNN (Eq. 39) also contains and edge operation and a node operation ϕ_e and ϕ_h re-

spectively. We use the same functions described above for both the GNN and the EGNN such that comparisons are as fair as possible.

B.3.1 *Implementation details for Dynamical Systems*

B.3.1.1 *Dataset*

In the dynamical systems experiment we used a modification of the Charged Particle's N-body (N=5) system from Kipf et al., 2018. Similarly to Fuchs et al., 2020, we extended it from 2 to 3 dimensions customizing the original code from (https://github.com/ethanfetaya/NRI) and we removed the virtual boxes that bound the particle's positions. The sampled dataset consists of 3.000 training trajectories, 2.000 for validation and 2.000 for testing. Each trajectory has a duration of 1.000 timesteps. To move away from the transient phase, we actually generated trajectories of 5.000 time steps and sliced them from timestep 3.000 to timestep 4.000 (1.000 time steps into the future) such that the initial conditions are more realistic than the Gaussian Noise initialization from which they are initialized.

In our second experiment, we sweep from 100 to 50.000 training samples, for this we just created a new training partition following the same procedure as before but now generating 50.000 trajectories instead. The validation and test partition remain the same from last experiment.

B.3.1.2 *Models*

All models are composed of 4 layers, the details for each model are the following.

EGNN: For the EGNN we use its variation that considers vector type inputs from Section 5.3.2. This variation adds the function ϕ_v to the model which is composed of two linear layers with one non-linearity: Input $\rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() } $\rightarrow$ Output. Functions ϕ_e, ϕ_x and ϕ_h that define our EGNN are the same than for all experiments and are described at the beginning of this Appendix B.3.

GNN: The GNN is also composed of 4 layers, its learnable functions edge operation ϕ_e and node operation ϕ_h from Equation 39 are exactly the same as ϕ_e and ϕ_h from our EGNN introduced in Appendix B.3. We chose the same functions for both models to ensure a fair comparison. In the GNN case, the initial position $\mathbf{P}^0$ and velocity $\mathbf{V}^0$ from the particles is passed through a linear layer and inputted into the GNN first layer $\mathbf{H}^0$. The particle's charges are inputted as edge attributes $\mathbf{a}_{ij} := c_i c_j$. The output of the GNN $\mathbf{H}^L$ is passed through a two layers MLP that maps it to the estimated position.

Radial Field: The Radial Field algorithm is described in the Related Work 5.4, its only parameters are contained in its edge operation $\phi_{rf}()$ which in our case is

a two layers MLP with two non linearities Input $\rightarrow$ {LinearLayer() $\rightarrow$ Swish() $\rightarrow$ LinearLayer() $\rightarrow$ Tanh } $\rightarrow$ Output. Notice we introduced a Tanh at the end of the MLP which fixes some instability issues that were causing this model to diverge in the dynamical system experiment. We also augmented the Radial Field algorithm with the vector type inputs modifications introduced in Section 5.3.2. In addition to the norms between pairs of points, $\phi_{rf}()$ also takes as input the particle charges $c_i c_j$.

Tensor Field Network: We used the Pytorch implementation from `https://github.com/FabianFuchsML/se3-transformer-public`. We swept over the parameters, degree $\in$ {2, 3, 4}, number of features $\in$ {12, 24, 32, 64, 128}. We got the best performance in our dataset for degree 2 and number of features 32. We used the Relu activation layer instead of the Swish for this model since it provided better performance.

SE(3) Transformers: For the SE(3)-Transformer we used code from `https://github.com/FabianFuchsML/se3-transformer-public`. Notice this implementation has only been validated in the QM9 dataset but it is the only available implementation of this model. We swept over different hyperparamters degree $\in$ {1, 2, 3, 4}, number of features $\in$ 16, 32, 64 and divergence $\in$ {1, 2}, along with the learning rate. We obtained the best performance for degree 3, number of features 64 and divergence 1. As in Tensor Field Networks we obtained better results by using the Relu activation layer instead of the Swish.

B.3.1.3 *Other implementation details*

In Table 6 all models were trained for 10.000 epochs, batch size 100, Adam optimizer, the learning rate was fixed and independently chosen for each model. All models are 4 layers deep and the number of training samples was set to 3.000.

B.3.2 *Implementation details for Graph Autoneoders*

B.3.2.1 *Dataset*

In this experiment we worked with Community Small (You et al., 2018) and Erdos&Renyi (Bollobás, 1998) generated datasets.

- Community Small: We used the original code from You et al., 2018 (`https://github.com/JiaxuanYou/graph-generation`) to generate a Community Small dataset. We sampled 5.000 training graphs, 500 for validation and 500 for testing.

- Erdos&Renyi is one of the most famous graph generative algorithms. We used the "gnp_random_graph(M, p)" function from (`https://networkx.org/`) that generates random graphs when povided with the number of nodes M

and the edge probability p following the Erdos&Renyi model. Again we generated 5.000 graphs for training, 500 for validation and 500 for testing. We set the edge probability (or sparsity value) to $p = 0.25$ and the number of nodes M ranging from 7 to 16 deterministically uniformly distributed. Notice that edges are generated stochastically with probability p, therefore, there is a chance that some nodes are left disconnected from the graph, "gnp_random_graph(M, p)" function discards these disconnected nodes such that even if we generate graphs setting parameters to $7 \leqslant M \leqslant 16$ and $p = 0.25$ the generated graphs may have less number of nodes.

Finally, in the graph autoencoding experiment we also overfitted in a small partition of 100 samples (Table 14) for the Erdos&Renyi graphs described above. We reported results for different p values ranging from 0.1 to 0.9. For each p value we generated a partition of 100 graphs with initial number of nodes between $7 \leqslant M \leqslant 16$ using the Erdos&Renyi generative model.

B.3.2.2 *Models*

All models consist of 4 layers, 64 features for the hidden layers and the Swish activation function as a non linearity. The EGNN is defined as explained in Section 5.3 without any additional modules (i.e. no velocity type features or inferring edges). The functions ϕ_e, ϕ_x and ϕ_h are defined at the beginning of this Appendix B.3. The GNN (Eq. 39) mimics the EGNN in terms that it uses the same ϕ_h and ϕ_e than the EGNN for its edge and node updates. The Noise-GNN is exactly the same as the GNN but inputting noise into the $\mathbf{H}_0$ features. Finally the Radial Field was defined in the Related Related work Section 5.4 which edge's operation ϕ_{rf} consists of a two layers MLP: Input $\rightarrow$ { Linear() $\rightarrow$ Swish() $\rightarrow$ Linear() } $\rightarrow$ Output.

B.3.2.3 *Other implementation details*

All experiments have been trained with learning rate 10^{-4}, batch size 1, Adam optimizer, weight decay 10^{-16}, 100 training epochs for the 5.000 samples sized datasets performing early stopping for the minimum Binary Cross Entropy loss in the validation partition. The overfitting experiments were trained for 10.000 epochs on the 100 samples subsets.

B.3.3 *Implementation details for QM9*

For QM9 Ramakrishnan et al., 2014 we used the dataset partitions from Anderson et al., 2019. We imported the dataloader from his code repository (`https://github.com/risilab/cormorant`) which includes his data-preprocessing. Additionally all properties have been normalized by subtracting the mean and dividing by the Mean Absolute Deviation.

Table 16: Analysis of the % of wrong edges and F1 score for different n embedding sizes {2, 4, 8 } for the GNN, Noise-GNN and EGNN in the Community Small dataset.

	Community Small					
	n=4		n=6		n=8	
	% Err.	F1	% Err.	F1	% Err.	F1
GNN	**1.45**	**0.977**	1.29	0.9800	1.29	0.980
Noise-GNN	1.94	0.970	0.44	0.9931	0.44	0.993
EGNN	2.19	0.966	**0.42**	**0.9934**	**0.06**	**0.999**

Our EGNN consists of 7 layers. Functions ϕ_e and ϕ_h are defined at the beginning of this Appendix B.3. Additionally, we use the module ϕ_{inf} presented in Section 5.3.3 that infers the edges . This function ϕ_{inf} is defined as a linear layer followed by a sigmoid: Input $\rightarrow$ {Linear() $\rightarrow$ sigmoid()} $\rightarrow$ Output. Finally, the output of our EGNN $\mathbf{h}^L$ is forwarded through a two layers MLP that acts node-wise, a sum pooling operation and another two layers MLP that maps the averaged embedding to the predicted property value, more formally: $\mathbf{h}^L \rightarrow$ {Linear() $\rightarrow$ Swish() $\rightarrow$ Linear() $\rightarrow$ Sum-Pooling() $\rightarrow$ Linear() $\rightarrow$ Swish() $\rightarrow$ Linear} $\rightarrow$ Property. The number of hidden features for all model hidden layers is 128.

We trained each property individually for a total of 1.000 epochs, we used Adam optimizer, batch size 96, weight decay 10^{-16}, and cosine decay for the learning rate starting at at a lr=$5 \cdot 10^{-4}$ except for the Homo, Lumo and Gap properties where its initial value was set to 10^{-3}.

B.4 FURTHER EXPERIMENTS

B.4.1 *Graph Autoencoder*

In this section we present an extension of the Graph Autoencoder experiment 5.5.2. In Table 16 and 17 we report the approximation error of the reconstructed graphs as the embedding dimensionality n is reduced $n \in \{4, 6, 8\}$ in the Community Small and Erdos&Renyi datasets for the GNN, Noise-GNN and EGNN models. For small embedding sizes ($n = 4$) all methods perform poorly, but as the embedding size grows our EGNN significantly outperforms the others.

Table 17: Analysis of the % of wrong edges and F1 score for different n embedding sizes {2, 4, 8 } for the GNN, Noise-GNN and EGNN in the Erdos&Renyi dataset.

	Erdos&Renyi					
	n=4		n=6		n=8	
	% Err.	F1	% Err.	F1	% Err.	F1
GNN	7.92	0.844	5.22	0.894	4.62	0.907
Noise-GNN	3.80	0.925	2.66	0.947	1.25	0.975
EGNN	**3.09**	**0.939**	**0.58**	**0.988**	**0.11**	**0.998**

B.5 SOMETIMES INVARIANT FEATURES ARE ALL YOU NEED

Perhaps surprisingly we find our EGNNs outperform other equivariant networks that consider higher-order representations. In this section we prove that when only positional information is given (i.e. no velocity-type features) then the geometry is completely defined by the invariant distance norms in-between points, without loss of relevant information. As a consequence, it is not necessary to consider higher-order representation types of the relative distances, not even the relative differences as vectors. To be precise, note that these invariant features still need to be *permutation* equivariant, they are only $\mathrm{E}(n)$ invariant.

To be specific, we want to show that for a collection of points $\{\mathbf{x}_i\}_{i=1}^M$ the norm of in-between distances $\ell_2(\mathbf{x}_i, \mathbf{x}_j)$ are a *unique* identifier of the geometry, where collections separated by an $\mathrm{E}(n)$ transformations are considered to be identical. We want to show *invariance* of the norms under $\mathrm{E}(n)$ transformations and *uniqueness*: two point collections are identical (up to $\mathrm{E}(n)$ transform) when they have the same distance norms.

Invariance. Let $\{\mathbf{x}_i\}$ be a collection of M points where $\mathbf{x}_i \in \mathbb{R}^n$ and the ℓ_2 distances are $\ell_2(\mathbf{x}_i, \mathbf{x}_j)$. We want to show that all $\ell_2(\mathbf{x}_i, \mathbf{x}_j)$ are unaffected by $\mathrm{E}(n)$ transformations.

Proof. Consider an arbitrary $\mathrm{E}(n)$ transformation $\mathbb{R}^n \to \mathbb{R}^n : \mathbf{x} \mapsto \mathbf{Q}\mathbf{x} + \mathbf{t}$ where Q is orthogonal and $\mathbf{t} \in \mathbb{R}^n$ is a translation. Then for all i, j:

$$\begin{aligned}
\ell_2(\mathbf{Q}\mathbf{x}_i + \mathbf{t}, \mathbf{Q}\mathbf{x}_j + \mathbf{t}) &= \sqrt{(\mathbf{Q}\mathbf{x}_i + \mathbf{t} - [\mathbf{Q}\mathbf{x}_j + \mathbf{t}])^T (\mathbf{Q}\mathbf{x}_i + \mathbf{t} - [\mathbf{Q}\mathbf{x}_j + \mathbf{t}])} \\
&= \sqrt{(\mathbf{Q}\mathbf{x}_i - \mathbf{Q}\mathbf{x}_j)^T (\mathbf{Q}\mathbf{x}_i - \mathbf{Q}\mathbf{x}_j)} \\
&= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{Q}^T \mathbf{Q} (\mathbf{x}_i - \mathbf{x}_j)} = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j)} = \ell_2(\mathbf{x}_i, \mathbf{x}_j)
\end{aligned}$$

This proves that the ℓ_2 distances are invariant under $\mathrm{E}(n)$ transforms.

Uniqueness. Let $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$ be two collection of M points each where all in-between distance norms are identical, meaning $\ell_2(\mathbf{x}_i, \mathbf{x}_j) = \ell_2(\mathbf{y}_i, \mathbf{y}_j)$. We want to show that $\mathbf{x}_i = \mathbf{Q}\mathbf{y}_i + \mathbf{t}$ for some orthogonal Q and translation $\mathbf{t}$, for all i.

Proof. Subtract $\mathbf{x}_0$ from all $\{\mathbf{x}_i\}$ and $\mathbf{y}_0$ from all $\{\mathbf{y}_i\}$, so $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{x}_0$ and $\tilde{\mathbf{y}}_i = \mathbf{y}_i - \mathbf{y}_0$. As proven above, since translation is an $E(n)$ transformation the distance norms are unaffected and:

$$\ell_2(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) = \ell_2(\mathbf{x}_i, \mathbf{x}_j) = \ell_2(\mathbf{y}_i, \mathbf{y}_j) = \ell_2(\tilde{\mathbf{y}}_i, \tilde{\mathbf{y}}_j).$$

So without loss of generality, we may assume that $\mathbf{x}_0 = \mathbf{y}_0 = \mathbf{0}$. As a direct consequence $\|\mathbf{x}_i\|_2 = \|\mathbf{y}_i\|_2$. Now writing out the square:

$$\mathbf{x}_i^T\mathbf{x}_i - 2\mathbf{x}_i^T\mathbf{x}_j + \mathbf{x}_j^T\mathbf{x}_j = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 = \mathbf{y}_i^T\mathbf{y}_i - 2\mathbf{y}_i^T\mathbf{y}_j + \mathbf{y}_j^T\mathbf{y}_j$$

And since $\|\mathbf{x}_i\|_2 = \|\mathbf{y}_i\|_2$, it follows that $\mathbf{x}_i^T\mathbf{x}_j = \mathbf{y}_i^T\mathbf{y}_j$ or equivalently written as dot product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \langle \mathbf{y}_i, \mathbf{y}_j \rangle$. Notice that this already shows that angles between pairs of points are the same.

At this moment, it might already be intuïtive that the collections of points are indeed identical. To finalize the proof formally we will construct a linear map A for which we will show that (1) it maps every $\mathbf{x}_i$ to $\mathbf{y}_i$ and (2) that it is orthogonal. First note that from the angle equality it follows immediately that for every linear combination:

$$\|\sum_i c_i\mathbf{x}_i\|_2 = \|\sum_i c_i\mathbf{y}_i\|_2 \quad (*).$$

Let V_x be the linear span of $\{\mathbf{x}_i\}$ (so V_x is the linear subspace of all linear combinations of $\{\mathbf{x}_i\}$). Let $\{\mathbf{x}_{i_j}\}_{j=1}^d$ be a basis of V_x, where $d \leqslant n$. Recall that one can define a linear map by choosing a basis, and then define for each basis vector where it maps to. Define a linear map A from V_x to V_y by the transformation from the basis $\mathbf{x}_{i_j}$ to $\mathbf{y}_{i_j}$ for $j = 1, ..., d$. Now pick any point $\mathbf{x}_i$ and write it in its basis $\mathbf{x}_i = \sum_j c_j\mathbf{x}_{i_j} \in V_x$. We want to show $A\mathbf{x}_i = \mathbf{y}_i$ or alternatively $\|\mathbf{y}_i - A\mathbf{x}_i\|_2 = 0$. Note that $A\mathbf{x}_i = A\sum_j c_j\mathbf{x}_{i_j} = \sum_j c_jA\mathbf{x}_{i_j} = \sum_j c_j\mathbf{y}_{i_j}$. Then:

$$\begin{aligned}\|\mathbf{y}_i - \sum_j c_j\mathbf{y}_{i_j}\|_2^2 &= \langle \mathbf{y}_i, \mathbf{y}_i \rangle - 2\langle \mathbf{y}_i, \sum_j c_i\mathbf{y}_{i_j} \rangle + \langle \sum_j c_i\mathbf{y}_{i_j}, \sum_j c_i\mathbf{y}_{i_j} \rangle \\ &\overset{(*)}{=} \langle \mathbf{x}_i, \mathbf{x}_i \rangle - 2\langle \mathbf{x}_i, \sum_j c_i\mathbf{x}_{i_j} \rangle + \langle \sum_j c_i\mathbf{x}_{i_j}, \sum_j c_i\mathbf{x}_{i_j} \rangle = \langle \mathbf{x}_i, \mathbf{x}_i \rangle - 2\langle \mathbf{x}_i, \mathbf{x}_i \rangle + \langle \mathbf{x}_i, \mathbf{x}_i \rangle = 0.\end{aligned}$$

Thus showing that $A\mathbf{x}_i = \mathbf{y}_i$ for all $i = 1, \ldots, M$, proving (1). Finally we want to show that A is orthogonal, when restricted to V_x. This follows since:

$$\langle A\mathbf{x}_{i_j}, A\mathbf{x}_{i_j} \rangle = \langle \mathbf{y}_{i_j}, \mathbf{y}_{i_j} \rangle = \langle \mathbf{x}_{i_j}, \mathbf{x}_{i_j} \rangle$$

for the basis elements $\mathbf{x}_{i_1}, ..., \mathbf{x}_{i_d}$. This implies that A is orthogonal (at least when restricted to V_x). Finally A can be extended via an orthogonal complement of V_x to the whole space. This concludes the proof for (2) and shows that A is indeed orthogonal.

C

E(N) EQUIVARIANT NORMALIZING FLOWS

C.1 EXPERIMENT DETAILS

In this Appendix section, we provide additional details about the implementation of the experiments. Initially, we define the neural networks used to formulate the dynamics of our proposed equivariant normalizing flow and the baselines used in the experiments. Specifically, we focus on E(n) Equivariant Flow dynamics (E-NF), Graph Normalizing Flow dynamics (GNF), Graph Normalizing Flow with attention (GNF-att), and Graph Normalizing Flow with attention and data augmentation (GNF-att-aug). In the following sections, we explicitly describe the dynamics used for the ENF, GNF, GNF-att, and GNF-att-aug baselines:

- E-NF (Dynamics): We can write the E-NF dynamics as the original EGNN 5.3 with the modification proposed in Section 6.5.2:

$$\mathbf{m}_{ij} = \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2\right) \tag{85}$$

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \sum_{j \neq i} \frac{(\mathbf{x}_i^l - \mathbf{x}_j^l)}{\|\mathbf{x}_i^l - \mathbf{x}_j^l\| + 1} \phi_x\left(\mathbf{m}_{ij}\right) \tag{86}$$

$$\mathbf{m}_i = \sum_{j \neq i} \phi_{inf}(\mathbf{m}_{ij}) \mathbf{m}_{ij}, \tag{87}$$

$$\mathbf{h}_i^{l+1} = \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right) \tag{88}$$

- GNF: The dynamics for this method are a standard Graph Neural Network (Section 2.3), which can also be interpreted as a variant of the EGNN without equivariance. The dataset coordinates $\mathbf{x}$ are treated as $\mathbf{h}$ features; therefore, they are provided as input to $\mathbf{h}^0$ through a linear mapping before its

first layer. Since our datasets do not consider adjacency matrices, we let $e_{ij} = 1$ for all ij.

$$\mathbf{m}_{ij} = \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l\right) \tag{89}$$

$$\mathbf{m}_i = \sum_{j \neq i} e_{ij}\mathbf{m}_{ij}, \tag{90}$$

$$\mathbf{h}_i^{l+1} = \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right) \tag{91}$$

- GNF-att: This model is almost identical to GNF, with the only difference being that it infers the edges through $e_{ij} = \phi_{inf}(\mathbf{m}_{ij})$, which can also be interpreted as a form of attention.

- GNF-att-aug: This is the exact same model as GNF-att. The only difference lies in the pre-processing of the data since we perform data augmentation by rotating the node positions before inputting them to the model.

The EGNN module described here consists of four Multilayer Perceptrons (MLPs): ϕ_e, ϕ_x, ϕ_h, and ϕ_{inf}. The GNF baseline is characterized by two MLPs, ϕ_e and ϕ_h, which are also present in the EGNN. Thus, we utilized the exact same architecture for these two functions. The design for all MLPs, with the exception of ϕ_x, was adopted from the original EGNN work (Satorras et al., 2021b). The descriptions of the modules are as follows:

- ϕ_e (edge operation): consists of a two-layer MLP with two SiLU (Nwankpa et al., 2018) activation functions that takes as input $(\mathbf{h}_i, \mathbf{h}_j, \|\mathbf{x}_i - \mathbf{x}_j\|^2)$ and outputs the edge embedding $\mathbf{m}_{ij}$.

- ϕ_x (coordinate operation): consists of a two layers MLP with a SiLU activation function in its hidden layer and a Tanh activation function at the output layer. It takes as input the edge embedding $\mathbf{m}_{ij}$ and outputs a scalar value.

- ϕ_h (node operation): consists of a two layers MLP with one SiLU activation function in its hidden layer and a residual connection: $[\mathbf{h}_i^l, \mathbf{m}_i] \rightarrow$ {Linear() $\rightarrow$ SiLU() $\rightarrow$ Linear() $\rightarrow$ Addition($\mathbf{h}_i^l$) } $\rightarrow \mathbf{h}_i^{l+1}$.

- ϕ_{inf} (edge inference operation): Connsists of a Linear layer followed by a Sigmoid layer that takes as input the edge embedding $\mathbf{m}_{ij}$ and outputs a scalar value.

C.2 DW4 AND LJ13 EXPERIMENTS

In the following we report the DW4 (equation 92) and LJ13 (equation 93) energy functions introduced in (Köhler et al., 2020a):

$$u^{\mathrm{DW}}(x) = \frac{1}{2\tau} \sum_{i,j} a\left(d_{ij} - d_0\right) + b\left(d_{ij} - d_0\right)^2 + c\left(d_{ij} - d_0\right)^4 \tag{92}$$

$$u^{\mathrm{LJ}}(x) = \frac{\epsilon}{2\tau} \left[\sum_{i,j} \left(\left(\frac{r_m}{d_{ij}} \right)^{12} - 2 \left(\frac{r_m}{d_{ij}} \right)^{6} \right) \right] \tag{93}$$

Where $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$ is the distance between two particles. The design parameters a, b, c, d and temperature τ from DW4 and the design parameters ϵ, r_m and τ from LJ13 are the same ones used in (Köhler et al., 2020a).

IMPLEMENTATION DETAILS All methods are trained with the Adam optimizer, weight decay 10^{-12}, batch size 100, the learning rate was tuned independently for each method which resulted in 10^{-3} for all methods except for the E-NF model which was $5 \cdot 10^{-4}$.

In tables 18 and 19 we report the same DW4 and LJ13 averaged results from Section 6.6.1 but including the standard deviations over the three runs.

C.3 QM9 POSITIONAL AND QM9

Both experiments QM9 and QM9 positional have been trained with batch size 128 and weight decay 10^{-12}. The learning rate was set to $5 \cdot 10^{-4}$ for all methods except for the E-NF and Simple dynamics where it was reduced to $2 \cdot 10^{-12}$.

Table 18: Negative Log Likelihood comparison on the test partition of DW4 dataset for different amount of training samples. Averaged over 3 runs and including standard deviations.

	DW4			
# Samples	10^2	10^3	10^4	10^5
GNF	11.93 ± 0.41	11.31 ± 0.07	10.38 ± 0.11	7.95 ± 0.17
GNF-att	11.65 ± 0.39	11.13 ± 0.38	9.34 ± 0.29	7.83 ± 0.15
GNF-att-aug	8.81 ± 0.23	8.31 ± 0.19	7.90 ± 0.04	7.61 ± 0.06
Simple dynamics	9.58 ± 0.05	9.51 ± 0.01	9.53 ± 0.02	9.47 ± 0.06
Kernel dynamics	8.74 ± 0.02	8.67 ± 0.01	8.42 ± 0.00	8.26 ± 0.03
E-NF	**8.31** ± 0.05	**8.15** ± 0.10	**7.69** ± 0.06	**7.48** ± 0.05

Table 19: Negative Log Likelihood comparison on the test partition of LJ13 dataset for different amount of training samples. Averaged over 3 runs and including standard deviations.

	LJ13			
# Samples	10	10^2	10^3	10^4
GNF	43.56 ± 0.79	42.84 ± 0.52	37.17 ± 1.79	36.49 ± 0.81
GNF-att	43.32 ± 0.20	36.22 ± 0.34	33.84 ± 1.60	32.65 ± 0.57
GNF-att-aug	41.09 ± 0.53	31.50 ± 0.35	30.74 ± 0.86	30.93 ± 0.73
Simple dynamics	33.67 ± 0.07	33.10 ± 0.10	32.79 ± 0.13	32.99 ± 0.11
Kernel dynamics	35.03 ± 0.48	31.49 ± 0.06	31.17 ± 0.05	31.25 ± 0.12
E-NF	**33.12** ± 0.85	**30.99** ± 0.95	**30.56** ± 0.35	**30.41** ± 0.16

Table 20: Neg. log-likelihood averaged over 3 passes, variance from dequantization and trace estimator.

	NLL
GNF-attention	-28.2 ± 0.49
GNF-attention-augmentation	-29.3 ± 0.02
E-NF (ours)	-59.7 ± 0.12

The flows trained on QM9 have all been trained for 30 epochs. Training these models takes approximately 2 weeks using two NVIDIA 1080Ti GPUs. The flows trained on QM9 Positional have been trained for 160 epochs in single NVIDIA 1080Ti GPUs. Simple Dynamics would train in less than a day, Kernel Dynamics around 2 days, the other methods can take up to 7 days. The training of the models becomes slower per epoch as the performance improves, due to the required steps in the ODE solver. For QM9, the model performance is averaged over 3 test set passes, where variance originates from dequantization and the trace estimator, see Table 20.

C.4 STABILITY OF MOLECULES BENCHMARK

In the QM9 experiment, we also report the % of stable molecules and atoms. This section explains how we test for molecule stability. In addition, we explain why there is not a set of rules that will judge *every* molecule in the dataset stable. First of all, we say that an atom is stable when the number of bonds with other atoms matches their valence. For the atoms used their respective valencies are (H: 1, C: 4, N: 3, O: 2, F: 1). A molecule is stable when *all* of its atoms are stable. The most straightforward method to decide whether atoms are bonded is to compare their relative distance. There are some limitations to this method. For instance,

QM9 contains snapshots of molecules in a single configuration, and in reality atoms of a molecule are constantly in motion. In addition, the type of molecule can also greatly influence the relative distances, for instance due to collisions, the Van der Waals force and inter-molecule Hydrogen bonds. Further, environmental circumstances such as pressure and temperature may also affect bond distance. For these reasons it is not possible to design a distance based rule that considers every molecule in QM9 stable, based only on a snapshot. To find the most optimal rules for QM9, we tune the average bond distance for every atom-type pair to achieve the highest molecule stability on QM9 on the train set with results in 95.3% stable molecules and 99% stable atoms. On the test set these rules result in 95.2% stable molecules and 99% stable atoms.

The specific distances that we used to define the types of bond (SINGLE, DOUBLE TRIPLE or NONE) are available in the code and were obtained from `http://www.wiredchemist.com/chemistry/data/bond_energies_lengths.html`. Notice the type of bond depends on the type of atoms that form that bond and the relative distance among them. Therefore, given a conformation of atoms, we deterministically compute the bonds among all pairs of atoms. Then we say an atom is stable if its number of bonds with other atoms matches its valence.

C.5 FURTHER QM9 ANALYSIS | VALIDITY, UNIQUENESS, NOVELTY

Validity is defined as the ratio of molecules that are valid from all the generated ones. *Uniqueness* is defined as the number of valid generated molecules that are unique divided by the number of all valid generated molecules. *Novelty* is defined as the number of valid generated molecules that are not part of the training set divided by the total number of valid generated molecules. In our case, all stable molecules are valid, therefore, we only report the Uniqueness and Novelty. Absolute and percentage values are reported when generating 10.000 examples. Metrics have been computed as in (Simonovsky and Komodakis, 2018) with rdkit `https://www.rdkit.org/`.

In addition to the previous metrics, in Figure 24, we plot a histogram of the number of atoms per molecule and also of the type of atoms for both the stable generated molecules and the ground truth ones.

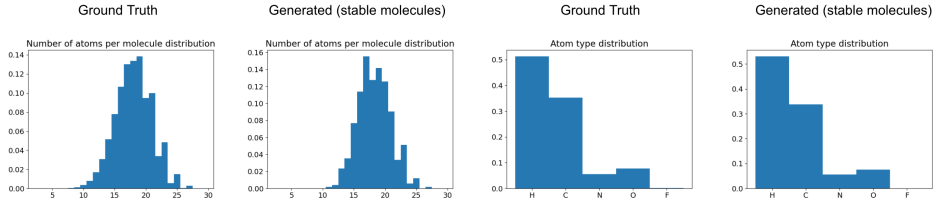

Figure 24: Number of atoms per molecule distributions and atom type distribution for the stable generated molecules and the training (ground truth) molecules.

D

EQUIVARIANT DIFFUSION FOR MOLECULE GENERATION IN 3D

D.1 ADDITIONAL DETAILS FOR THE METHOD

D.1.1 *The Dynamics*

In Section 7.4.4 we explained that the dynamics of our proposed Equivariant Diffusion Model (EDM) are learned by the EGNN introduced in Section 7.3.4. The EGNN consists of a sequence of Equivariant Graph Convolutional Layers (EGCL). The EGCL is defined in Eq. 65. All its learnable components ϕ_e, ϕ_h, ϕ_x, ϕ_{inf} by Multilayer Perceptrons:

Edge operation ϕ_e. Takes as input two node embeddings. The squared distance $d_{ij}^2 = \|\mathbf{x}_i^l - \mathbf{x}_j^l\|_2^2$, and the squared distance at the first layer as the optional attribute $a_{ij} = \|\mathbf{x}_i^0 - \mathbf{x}_j^0\|_2^2$ and outputs $\mathbf{m}_{ij} \in \mathbb{R}^{nf}$.
$\text{concat}[\mathbf{h}_i^l, \mathbf{h}_j^l, d_{ij}^2, \mathbf{a}_{ij}] \rightarrow \{\text{Linear}(nf \cdot 2 + 2, nf) \rightarrow \text{Silu} \rightarrow \text{Linear}(nf, nf) \rightarrow \text{Silu}\} \rightarrow \mathbf{m}_{ij}$

Edge inference operation ϕ_{inf}. Takes as input the message $\mathbf{m}_{ij}$ and outputs a scalar value $\tilde{e}_{ij} \in (0, 1)$.
$\mathbf{m}_{ij} \rightarrow \{\text{Linear}(nf, 1) \rightarrow \text{Sigmoid}\} \rightarrow \tilde{e}_{ij}$

Node update ϕ_h Takes as input a node embedding and the aggregated messages and outputs the updated node embedding.
$\text{concat}[\mathbf{h}_i^l, \mathbf{m}_{ij}] \rightarrow \{\text{Linear}(nf \cdot 2, nf) \rightarrow \text{Silu} \rightarrow \text{Linear}(nf, nf) \rightarrow \text{add}(\cdot, \mathbf{h}_i^l)\} \rightarrow \mathbf{h}_i^{l+1}$

Coordinate update ϕ_x. Has the same inputs as ϕ_e and outptus a scalar value.
$\text{concat}[\mathbf{h}_i^l, \mathbf{h}_j^l, d_{ij}^2, a_{ij}] \rightarrow \{\text{Linear}(nf \cdot 2 + 2, nf) \rightarrow \text{Silu} \rightarrow \text{Linear}(nf, nf) \rightarrow \text{Silu} \rightarrow \text{Linear}(nf, 1)\} \rightarrow \text{Output}$

D.1.2 *Baseline model*

While our EDM model is parametrized by an $E(3)$ equivariant EGNN network, the GDM model used for the ablation study uses a non equivariant graph network. In this network, the coordinates are simply concatenated with the other node features: $\tilde{\mathbf{h}}_i^0 = [\mathbf{x}_i, \mathbf{h}]$. A message passing neural network (Gilmer et al., 2017) is then applied, that can be written:

$$\tilde{\mathbf{h}}_i^{l+1} = \phi_h(\tilde{\mathbf{h}}_i^l, \sum_{j \neq i} \tilde{e}_{ij} \mathbf{m}_{ij}) \qquad \text{for} \quad \mathbf{m}_{ij} = \phi_e\left(\tilde{\mathbf{h}}_i^l, \tilde{\mathbf{h}}_j^l, a_{ij}\right)$$

The MLPs ϕ_e, ϕ_h are parametrized in the same way as in EGNN, with the sole exception that the input dimension of ϕ_e in the first layer is changed to accommodate the atom coordinates.

D.2 ADDITIONAL DETAILS ON EXPERIMENTS

D.2.1 *QM9*

On QM9, the EDM and GDMs are trained using EGNNs with 256 hidden features and 9 layers. The models are trained for 1100 epochs, which is around 1.7 million iterations with a batch size of 64. The models are saved every 20 epochs when the validation loss is lower than the previously obtained number. The diffusion process uses $T = 1000$. Training takes approximately 7 days on a single NVIDIA GeForce GTX 1080Ti GPU. When generating samples the model takes on average 1.7 seconds per sample on the 1080Ti GPU. The EDM that only models heavy atoms and no hydrogens has the same architecture but is faster to train because it operates over less nodes: it takes about 3.2 days on a single 1080Ti GPU for 1100 epochs and converges even earlier to its final performance.

D.2.2 *GEOM-DRUGS*

On GEOM, the EDM and GDMs are trained using EGNNs with 256 hidden features and 4 layers. The models are trained for 13 epochs, which is around 1.2 million iterations with a batch size of 64. Training takes approximately 5.5 days on three NVIDIA RTX A6000 GPUs. The model then takes on average 10.3 seconds to generate a sample.

BOND DISTANCES In order to check the validity and stability of the generated structures, we compute the distance between all pairs of atoms and use these distances to predict the existence of bonds and their order. Bond distances in Table 21, 22 and 23 are based on typical distances in chemistry[1][2]. In addition,

1 http://www.wiredchemist.com/chemistry/data/bond_energies_lengths.html
2 http://chemistry-reference.com/tables/Bond%20Lengths%20and%20Enthalpies.pdf

margins are defined for single, double, triple bonds $m_1, m_2, m_3 = 10, 5, 3$ which were found empirically to describe the QM9 dataset well. If an two atoms have a distance shorter than the typical bond length plus the margin for the respective bond type, the atoms are considered to have a bond between them. The allowed number of bonds per atom are: H: 1, C: 4, N: 3, O: 2, F: 1, B: 3, Al: 3, Si: 4, P: [3, 5], S: 4, Cl: 1, As: 3, Br: 1, I: 1. After all bonds have been created, we say that an atom is stable if its valency is precisely equal to the allowed number of bonds. An entire molecule is considered stable if *all* its atoms are stable. Although this metric does not take into account more atypical distances or aromatic bonds, it is still an extremely important metric as it measures whether the model is positioning the atoms precisely enough. On the QM9 dataset it still considers 95.2% molecules stable and 99.0% of atoms stable. For Geom-Drugs the molecules are much larger which introduces more atypical behaviour. Here the atom stability, which is 86.5%, can still be used since it describes how many atoms satisfy the typical bond length description. However, the molecule stability is 2.8% on the dataset, which is too low to draw meaningful conclusions.

Table 21: Typical bond distances for a single bond.

	H	C	O	N	P	S	F	Si	Cl	Br	I	B	As
H	74	109	96	101	144	134	92	148	127	141	161	119	152
C	109	154	143	147	184	182	135	185	177	194	214	–	–
O	96	143	148	140	163	151	142	163	164	172	194	–	–
N	101	147	140	145	177	168	136	–	175	214	222	–	–
P	144	184	163	177	221	210	156	–	203	222	–	–	–
S	134	182	151	168	210	204	158	200	207	225	234	–	–
F	92	135	142	136	156	158	142	160	166	178	187	–	–
Si	148	185	163	–	–	200	160	233	202	215	243	–	–
Cl	127	177	164	175	203	207	166	202	199	214	–	175	–
Br	141	194	172	214	222	225	178	215	214	228	–	–	–
I	161	214	194	222	–	234	187	243	–	–	266	–	–
B	119	–	–	–	–	–	–	–	175	–	–	–	–
As	152	–	–	–	–	–	–	–	–	–	–	–	–

Table 22: Typical bond distances for a double bond.

	C	O	N	P	S
C	134	120	129	–	160
O	120	121	121	150	–
N	129	121	125	–	–
P	–	150	–	–	186
S	–	–	–	186	–

Table 23: Typical bond distances for a triple bond.

	C	O	N
C	120	113	116
O	113	–	–
N	116	–	110

D.3 SAMPLES FROM OUR MODELS

Additional samples from the model trained on QM9 are depicted in Figure 25 and, and samples from the model trained on GEOM-DRUGS in Figure 26. These samples are not curated or cherry picked in any way. As a result, their structure may sometimes be difficult to see due to an unfortunate viewing angle.

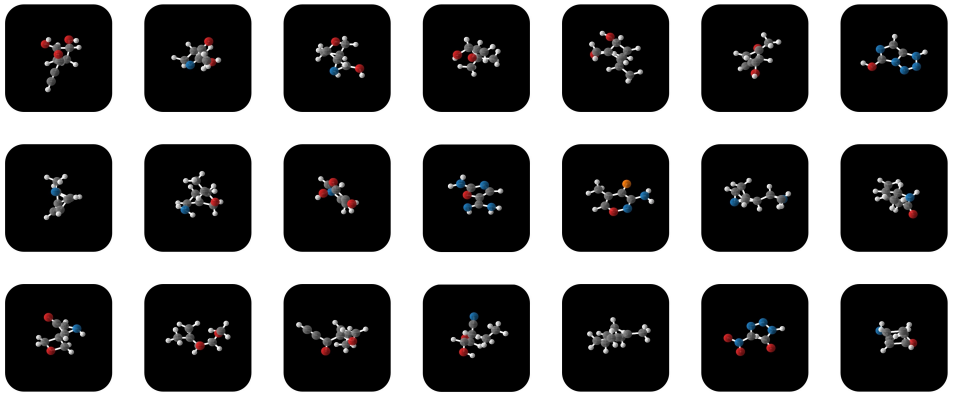

Figure 25: Random samples taken from the EDM trained on QM9.

The samples from the model trained on the drugs partition of GEOM show impressive large 3D structures. Interestingly, the model is sometimes generating disconnected component, which only happens QM9 models in early training stages. This may indicate that further training and increasing expressitivity of the models may further help the model bring these components together.

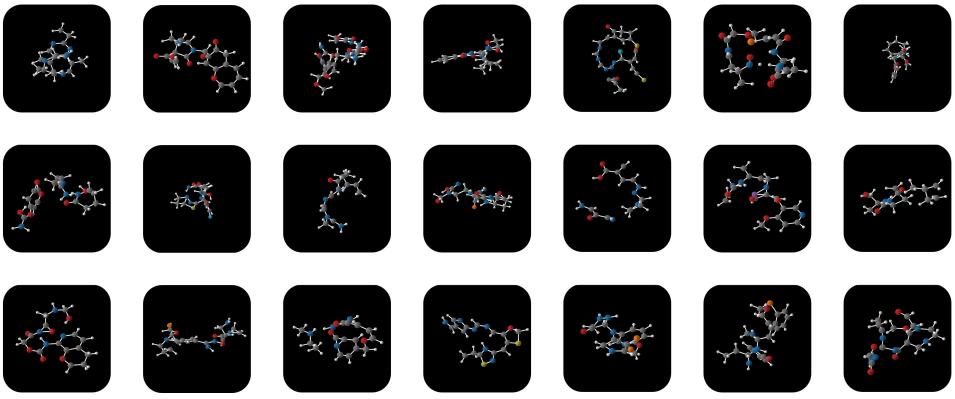

Figure 26: Random samples taken from the EDM trained on geom drugs. While most samples are very realistic, we observe two main failure cases: some molecules that are disconnected, and some that contain long rings. We note that the model does not feature any regularization to prevent these phenomena.

Figure 27 depicts the generation of molecules from a model trained on GEOM-Drugs. The model starts at random normal noise at time $t = T = 1000$ and iteratively sample $\mathbf{z}(t-1) \sim p(\mathbf{z}(t-1)|\mathbf{z}(t))$ towards $t = 0$ to obtain $\mathbf{x}, \mathbf{h}$, which is the resulting sample from the model. The atom type part of $\mathbf{z}(t)^{(h)}$ is visualized by taking the argmax of this component.

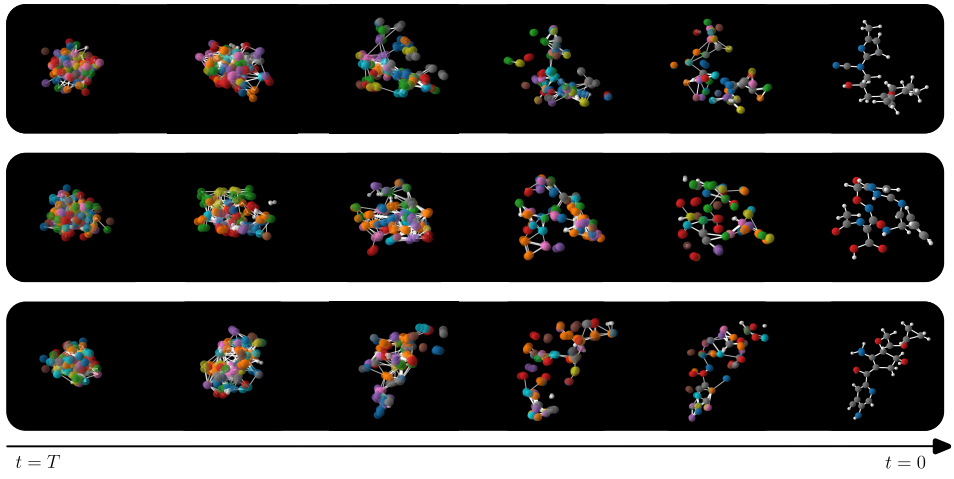

Figure 27: Selection of sampling chains at different steps from a model trained on GEOM-Drugs. The final column shows the resulting sample from the model.

D.4 ABLATION ON SCALING FEATURES

In Table 24 a comparison between the standard and proposed scaling is shown. Interestingly, there is quite a large difficulty in performance when measuring atom and molecule stability. From these results, it seems that it is easier to learn a denoising process where the atom type is decided later, when the atom coordinates are already relatively well defined.

Table 24: Ablation study on the scaling of features of the EDM. Comparing our proposed scaling to no scaling.

# Metrics	Scaling	NLL	Atom stable (%)	Mol stable (%)
EDM (ours)	$[\mathbf{x}, 1.00\ \mathbf{h}^{\text{onehot}}, 1.0\ \mathbf{h}^{\text{atom charge}}]$	-103.4	95.7	46.9
EDM (ours)	$[\mathbf{x}, 0.25\ \mathbf{h}^{\text{onehot}}, 0.1\ \mathbf{h}^{\text{atom charge}}]$	$\mathbf{-110.7}_{\pm 1.5}$	$\mathbf{98.7}_{\pm 0.1}$	$\mathbf{82.0}_{\pm 0.4}$
Data			99.0	95.2

D.5 CONDITIONAL GENERATION

D.5.1 *Conditional Method*

The specific definition for the loss components $\mathcal{L}_{c,t}$ is given in Equation 94. Essentially, a conditioning on a property c is added where relevant. The diffusion process that adds noise is not altered. The generative denoising process is conditioned on c by adding it as input to the neural network ϕ:

$$\begin{aligned}
\mathcal{L}_{c,t} &= \mathbb{E}_{\epsilon_t \sim \mathcal{N}_{x\,h}(0,\mathbf{I})}\big[\frac{1}{2}(1 - \text{SNR}(t-1)/\text{SNR}(t))\|\epsilon_t - \phi(\mathbf{z}(t), t, c)\|^2\big], \\
\mathcal{L}_{c,0}^{(h)} &= \log p(\mathbf{h}|\mathbf{z}(0)^{(h)}) \approx 0, \\
\mathcal{L}_{c,0}^{(x)} &= \log p(\mathbf{x}|\mathbf{z}(0), c) = \mathbb{E}_{\epsilon^{(x)} \sim \mathcal{N}(0,\mathbf{I})}\Big[\log Z^{-1} - \frac{1}{2}\|\epsilon^{(x)} - \phi^{(x)}(\mathbf{z}(0), 0, c)\|^2\Big], \\
\mathcal{L}_{c,\text{base}} &= \mathcal{L}_{\text{base}} = -\text{KL}(q(\mathbf{z}(T)|\mathbf{x}, \mathbf{h})|p(\mathbf{z}(T))) \approx 0.
\end{aligned} \tag{94}$$

Given a trained conditional model $p(\mathbf{x}, \mathbf{h}|c, M)$, we define the generative process by first sampling $c, M \sim p(c, M)$ and then $\mathbf{x}, \mathbf{h} \sim p(\mathbf{x}, \mathbf{h}|c, M)$. We compute $c, M \sim p(c, M)$ on the training partition as a parametrized two dimensional categorical distribution where we discretize the continuous variable c into small uniformly distributed intervals.

D.5.2 *Implementation details:*

In this conditional experiment, our Equivariant Diffusion Model uses an EGNN with 9 layers, 192 features per hidden layer and SiLU activation functions. We used the Adam optimizer with learning rate 10^{-4} and batch size 64. Only atom types (categorical) and positions (continuous) have been modelled but not atom charges. All methods have been trained for $\sim$ 2000 epochs while doing early stopping by evaluating the Negative Log Likelihood on the validation partition proposed by (Anderson et al., 2019).

Additionaly, the obtained molecule stabilities in the conditional generative case was similar to the the ones obtained in the non-conditional setting. The reported molecule stabilities for each conditioned property evaluated on 10K generated samples are: (80.4%) α, (81.73%) $\Delta\varepsilon$, (82.81%) $\varepsilon_{\text{HOMO}}$, (83.6 %) $\varepsilon_{\text{LUMO}}$, (83.3%) μ, (81.03 %) C_ν.

D.5.3 *QM9 Properties*

α Polarizability: Tendency of a molecule to acquire an electric dipole moment when subjected to anexternal electric field.

$\varepsilon_{\text{HOMO}}$: Highest occupied molecular orbital energy.

$\varepsilon_{\text{LUMO}}$: Lowest unoccupied molecular orbital energy.

$\Delta\varepsilon$ Gap: The energy difference between HOMO and LUMO.

μ: Dipole moment.

C_ν: Heat capacity at 298.15K

D.5.4 *Conditional generation results*

In this Section we sweep over 9 different α values in the range [73.6, 101.6] while keeping the reparametrization noise ϵ fixed and the number of nodes $M = 19$. We plot 10 randomly selected sweeps in Figure 28 with different reparametrization noises ϵ each. Samples have been generated using our Conditional EDM. We can see that for larger Polarizability values, the atoms are distributed less isotropically encouraing larger dipole moments when an electric field is applied. This behavior is consistent among all reported runs.

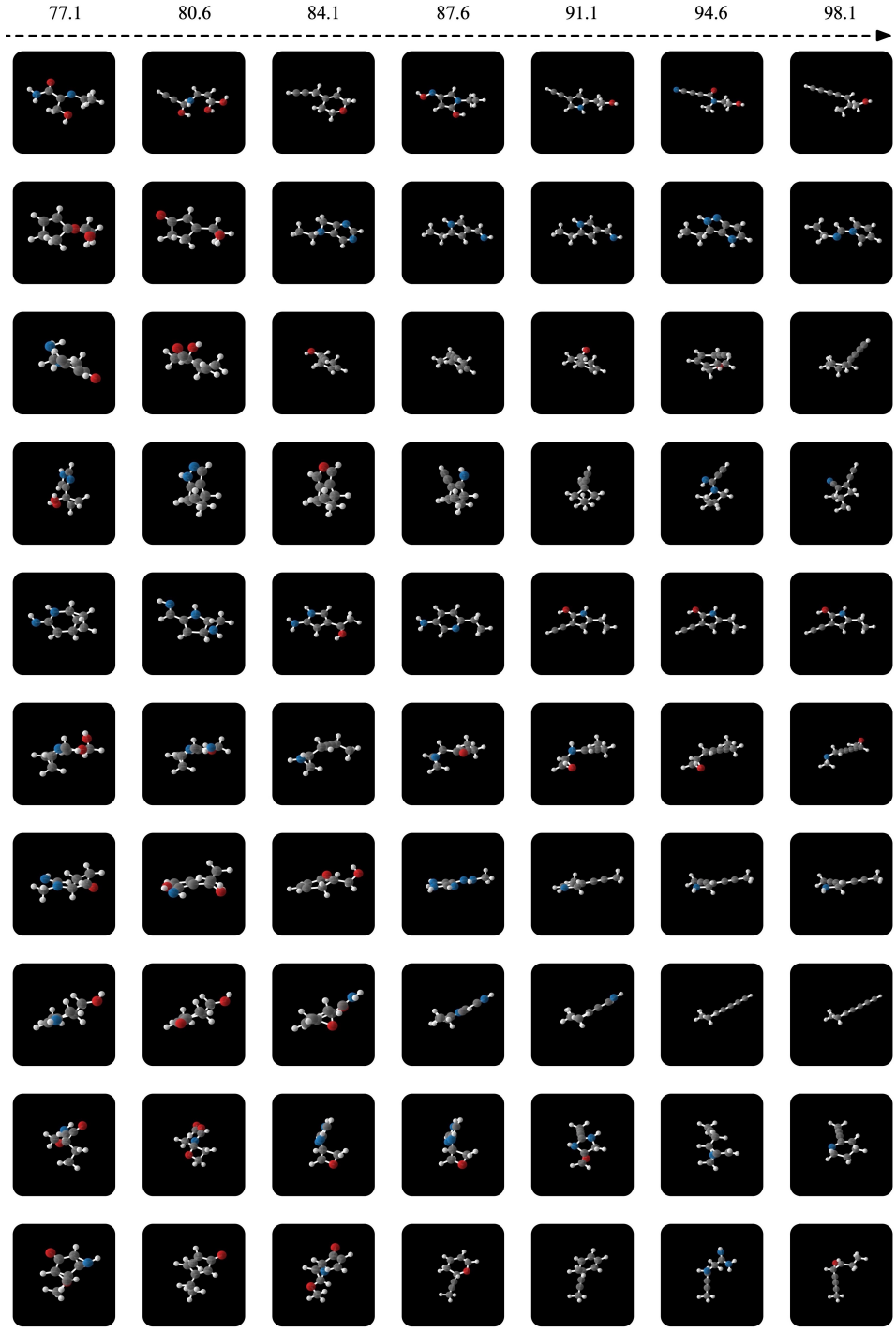

Figure 28: Molecules generated by our Conditional EDM when interpolating among different α polarizability values (from left to right). α's are reported on top of the image. All samples within each row have been generated with the same reparametrization noise ϵ.