SE(3) Equivariant Neural Networks for Regression on Molecular Properties

The QM9 Benchmark

Benjamin Kurt Miller

A thesis presented for the degree of
Masters of Science (M.Sc.) in Computational Science

Supervised by
Prof. Dr. Frank Noé

Computational Sciences - Molecular Science
Freie Universität Berlin
Berlin, Germany
31 March 2020

Colloquium held on
17 March 2020

Contents

Abstract **1**

Declaration of Authorship **2**

Acknowledgments **3**

1 Introduction **6**

2 Theory **7**

2.1 Machine Learning and Deep Learning . . . 7

2.1.1 The Bias-Variance Decomposition . . . 8

2.1.2 Training and Optimization of the Loss Function . . . 11

2.2 The QM9 Data Set . . . 12

2.2.1 Quantum Mechanical Calculations . . . 14

2.3 Geometric Quantities . . . 16

2.3.1 Groups and Representations . . . 17

2.3.2 Group Representation . . . 18

2.3.3 The Spherical Harmonics and the Wigner D-Matrices . . . 19

2.3.4 The Clebsch-Gordan Decomposition . . . 20

2.4 Neural Networks Equivariant to SE(3) . . . 21

2.4.1 e3nn . . . 21

2.4.2 QM9 Output Layer . . . 29

3 Experiments **32**

3.1 Comparison with SchNet-like Architecture . . . 32

3.1.1 Effects of Higher Rotation Orders on Dipole Moment and Energy . . 36

3.1.2 Comparison of Radial Bases . . . 37

3.1.3 Effects of Batch Size . . . 37

3.2 e3nn with a Multi-Layer Perceptron Output . . . 38

3.2.1 Multi-target Hyperparameter Search . . . 40

3.3 Regression on QM9 Properties . . . 42

3.3.1 Dipole Moment . . . 42

3.3.2 Isotropic Polarizability . . . 44

3.3.3 Other Targets . . . 46

4 Conclusion **47**

A Definitions of Common Operations **48**

B Kernel Normalization Constants **49**

Abstract

e3nn is an artificial neural network which operates on atomic coordinates and achieves equivariance to the special euclidean group in three dimensions by using spherical harmonics as features. The main experiment is to benchmark the model against a standard chemical data set called QM9, on which e3nn achieves state of the art performance on three of twelve regression targets. Along with empirical results, this thesis presents theoretical argumentation for why e3nn outperforms its closest relatives, SchNet and Cormorant, on some regression targets. Significant background regarding machine learning, quantum chemistry, and the special euclidean group is also presented.

Declaration of Authorship

I, Benjamin Kurt Miller, declare that this thesis and the work presented in it is my own.

This work was completed while in candidature for a master's degree in Computational Science at the Freie Universität Berlin. Whenever I have quoted from the work of others, the source is always given and attributed.

Signed:

Date:

Acknowledgments

Learning about neural networks and the symmetries of Euclidean space has been a long and winding road. I would like to acknowledge the people who helped me along the path. I'll start with my introduction to the subject through my advisor, Dr. Frank Noé, and his PhD student, Moritz Hoffmann. My understanding of the subject was strengthened thanks to the countless hours of discussion and problem solving during my summer at Lawrence Berkeley National Laboratory where I worked closely with Dr. Tess Smidt (Berkeley Lab) and Mario Geiger (EPFL). Upon return to the Freie Universität Berlin, the days in the office were made much more enjoyable by discussing machine learning and physics with Zeno Schätzle. I'll remember fondly the "Mittagstisch" lunch group meetings of Eva, Tracy, and myself. I acknowledge my colleagues from the entire Artificial Intelligence for the Sciences group as well as the powerful deep learning tools which make this research possible, TensorFlow [1] and PyTorch [2].

On the personal side of things, I appreciated the support of my family: Gretchen, Doug, Abby, and Jeffrey. Finally, I'd like to dedicate this work to my friends and family who were there at the beginning but did not make it to the finish: Asher, Katerina "Katy," Katharine "Grandma Kay," and Lea.

Preamble: Why Data-Driven Methods? Why Deep Learning?

Low-latency, high-capacity, and low-cost storage methods along with pervasive access to fast internet connections has contributed to the proliferation of high-quality, easily-distributed data. Navigating these large data sets has become a field unto itself and competency in data science has emerged as a necessary skill for researchers of all backgrounds. One of the great data-related challenges lies in drawing insights from sprawling landscapes of tabulated strings, floats, and integers. Much like the adage, “Can’t see the forest for the trees,” large data sets can obscure knowledge by drowning trends in overwhelmingly large numbers of examples or embedding low-dimensional information in a prohibitively high-dimensional space, i.e. including extraneous information. An effective tool in the landscape of data-driven methods for transforming data into insights is called machine learning.

The 2010s were characterized by a groundswell of interest in the development of algorithms capable of learning from large quantities of data automatically. In particular, deep learning has enjoyed profound development due to a cocktail of factors including commercial success, prevalence of highly parallel graphics processing units, easy-to-use modular implementations of deep learning algorithms, and deep learning’s effectiveness in the regime of large data sets. Deep learning helped overcome barriers in language translation [3] and object classification in images [4] while also introducing groundbreaking techniques in generative modeling including the Generative Adversarial Network [5] and the Variational Auto-encoder [6]. Generative models have made it possible to draw samples from complicated distributions in physics like with Boltzmann Generators [7] but also from distributions of pixels which look like cats, dogs, and even celebrities [8].

The success of deep learning in computer science has already inspired active research efforts and a rich set of literature about using deep learning in chemistry and physics [9, 10]. The output of computational methods like density functional theory inadvertently produce data sets which complement machine learning by providing labeled data for regression. A common chemical benchmark data set, QM9 [11, 12], is a perfect example of the interplay between computational chemistry and data-driven methods since the targets were calculated using density functional theory and have been tested extensively with a wide variety of deep learning methods. In the future, the author predicts that existing attempts to explore chemical space, the term for the set of all possible chemicals, using high-throughput screening pipelines will be significantly enhanced by the reduced computational cost of using trained deep learning models.

The evidence of laboratory-confirmed effectiveness of machine learning methods to identify candidate molecules is growing. Notably, the message-passing molecular graph model MoleculeNet [13] has recently been critical in the discovery of a novel, experimentally-effective, and low human toxicity antibiotic called Halicin [14]. (Note the name was chosen as a reference to the 2001: A Space Odyssey AI character.) Previous methods to identify candidate antibiotics utilized a representation method known as fingerprint vectors, which reflected the presence or absence of functional groups in the molecule and included other computed molecular properties. Expert knowledge was required to create these fingerprint vectors and even after years of development, such methods were not very effective [15]. In comparison, deep learning has reduced the expert knowledge required to compute which molecules are strong antibiotic candidates and increased the speed at which these candidates can be discovered.

The practical motivation for this work is to enhance the computational options of researchers looking to understand and perhaps design chemicals using a data-driven approach. The experimental focus is on regression and the value of symmetry aware neural networks for reducing generalization error. The primary experiment is on benchmarking QM9 with a neural network equivariant to special euclidean symmetry. How does it help researchers? The regression techniques explored in Chapter 3 represent a method similar to the antibiotic selection method above, i.e. creating a neural network which can quickly and accurately predict geometric quantities in order to sift through massive databases of molecular data to find drug candidates.

What distinguishes this work from another prominent work in the field, SchNet [16], is the focus on the connection between molecular properties and Euclidean geometry. The argumentation is that, in order to accurately model properties that rotate, such as geometric tensors, the internal features of the neural network must have the appropriate rotation properties. This is achieved by using an architecture called e3nn [17]. It has a similar design to another network called Cormorant [18], but includes gated nonlinearities which empirically lead to better generalization on the QM9 benchmark. e3nn achieves state of the art results on three of twelve targets while performing adequately on the rest.

Chapter 1

Introduction

Calculating the properties of molecules on the computer can lead to a better understanding of chemistry, the discovery of new and useful chemical compounds, and the advancement of physical approximation methods. The current workhorse of this field is density functional theory which approximates the quantum mechanical properties of a molecule using a combination of theoretically motivated and empirically tested tools. Right now, if a scientist wants to understand the properties of a small to medium sized molecule, the tool they usually reach for is density functional theory. For smaller molecules or more accurate predictions, fully quantum theories are utilized including Hartree-Fock and Coupled Cluster. For larger molecules, empirically determined force field models with fast calculation times act as a cost effective approximation because quantum mechanical effects have less influence at the macroscopic scale.

There is a fundamental friction between seeking higher accuracy calculations and keeping computational costs low, even for small molecules. Chemical calculation complexity usually scales with the number of atoms and higher accuracy often leads to worse scaling. The ambition of current research is to increase accuracy while maintaining, or even reducing, computational costs. Machine learning has offered a solution by dividing the computational cost into a rather slow training procedure followed by a blazingly fast inference phase.

There have already been significant efforts made in this field [16, 19, 20] with strong results. However, the author critiques that these models, with the exception of [18], do not take the underlying symmetries of the problem into account in a satisfactory way. Could we improve the performance of these models by considering the symmetries of the problem?

This work investigates the performance of e3nn on a standard data set called QM9. e3nn is an artificial neural network which restricts itself to operations which are equivariant to the special euclidean group in three dimensions, which is to say, it handles rotations and translations of input data naturally [17]. This is a very desirable property when the input data are positions of constituent atoms which have no canonical orientation.

The content is broken into a theoretical section and an experiment section. The theoretical section covers deep learning; the benchmark data set, including how it was calculated; the basics of the special euclidean group; and finally explains how e3nn works. The experiment section discusses the specifics of how accurately e3nn was able to predict the targets of QM9. The performance of e3nn is systematically evaluated across several hyperparameters: the rotation properties of the network, the radial model, and the batch size. Finally, a random hyperparameter search yields a competitive e3nn model which is compared with other literature.

Chapter 2

Theory

2.1 Machine Learning and Deep Learning

The field of machine learning has enjoyed an explosion of interest and development in the 2010s. For the curious reader, three strong textbooks regarding supervised and unsupervised learning include Bishop [21], Murphy [22], and Goodfellow, Bengio, and Courville [23].

The purpose of machine learning is to automatically extract useful patterns in a set of training data which generalize well to another, unseen, set of data. The key components are the machine learning algorithm used to identify those patterns, the nature of the data itself, and this idea of generalization. A popular example machine learning data set called MNIST [24] will be included to help guide the discussion.

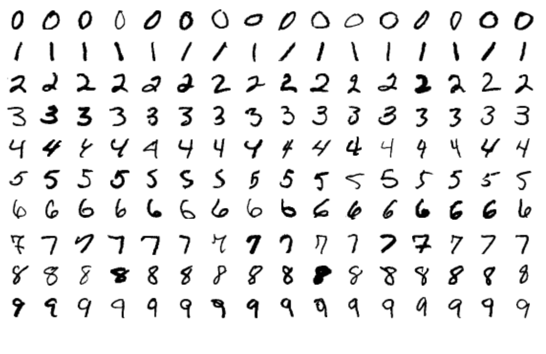

Figure 2.1: The samples of MNIST are hand-written digits from United States Census Bureau employees and high school students. The target vector is pair-wise identical between images in the same row. Samples from MNIST. Grid organized by Wikipedia.org.

Consider a set of vector pairs called training data $D_i = \{(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_N, \mathbf{t}_N)\}$ with $\mathbf{x} \in X$ and $\mathbf{t} \in T$. Each input $\mathbf{x}_i$ is associated to its pair, the target $\mathbf{t}_i$. The form of the inputs and targets can vary significantly between problems. In MNIST we define X to be the set of black and white images which contain a hand written digit, $X \coloneqq \{\mathbf{x} \in \mathbb{R}^{28\times 28} : \mathbf{x} \text{ is a hand written digit}\}$ and the set of one-hot scalar array targets T is defined as $T \coloneqq \left\{ v \in \{0,1\}^n : \sum_{i=1}^{n} v_i = 1 \right\}$. Examples of input data can be seen in Figure 2.1.

The machine learning algorithm can be written as a function $f : X \times W \to Y$. The specifics of the function, i.e. the parameters $\mathbf{w} \in W$ are determined during training and fixed afterwards during evaluation. How the parameters are chosen is further discussed in Section 2.1.2, but the idea is to select parameters which perform well on data which was drawn from the same distribution as the training set but was **not** included in the training set, also known as test data. When an algorithm with trained parameters performs well on test data we say that the algorithm generalizes well. In our MNIST example a trained algorithm f with parameters $\mathbf{w}$ generalizes well when $f(\tilde{\mathbf{x}}, \mathbf{w}) \approx \tilde{\mathbf{t}}$ for all $\left\{(\tilde{\mathbf{x}}, \tilde{\mathbf{t}}) \in (X, T) : (\tilde{\mathbf{x}}, \tilde{\mathbf{t}}) \notin D\right\}$.

The class of problems where both input and target data enter the training is known as supervised learning. The goal of the MNIST problem is to successfully recognize and categorize each sample into a finite number of discrete categories. Settings of this type are called classification problems. Another common supervised learning problem aims to predict a continuous target vector. This setting is known as regression and is utilized in this work.

2.1.1 The Bias-Variance Decomposition

Although there is some contention on the issue of generalization and overfitting regarding neural networks [25], an important frequentist perspective on supervised learning in general is the so-called bias-variance trade-off [21]. A set of predictive models has the property that models with lower estimation bias exhibit higher variance across samples, and vice versa. Bias measures the average distance between the predictions and the ground truth while variance measures the average distance between predictions. Consider figure 2.2 in order to gain a better intuition regarding bias and variance.

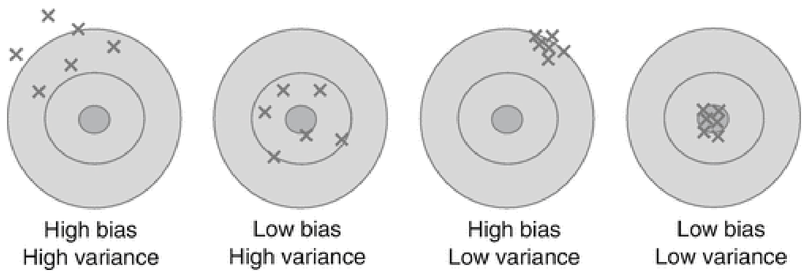

Figure 2.2: In this dartboard graphic from [26], the bullseye represents a ground truth regression target $f(x)$. The crosses correspond to predictions from multiple arbitrary estimators, $\hat{f}$, trained on different data sets of a fixed size to regress on $f(x)$. The labels below describe the bias and variance properties of the model. Bias describes the distance between the mean prediction and the bullseye while variance measures how far apart the crosses are on average. Lower bias and lower variance is desirable since predictions of such models are closer to the ground truth; however, due to the bias-variance trade-off, this is not always possible to achieve in practice.

The bias-variance trade-off is a dilemma where attempting to minimize both the variance and bias at the same time is in conflict and limits the ability of the model to generalize. This property is inherent no matter the choice of loss function and can be demonstrated using the bias–variance decomposition whereby the loss is analyzed as the sum of bias, variance, and aleatoric (inherent or irreducible) terms. For this analysis we consider the squared loss function in order to quantify the “wrongness” of a predicted value. The bias–variance decomposition of the squared loss function is presented below. It closely follows the derivation in [21] but has been summarized and clarified.

As this is a frequentist perspective we assume there is a deterministic function f mapping from input data x to the target space plus noise, i.e. define the target random variable

$T = f(x) + \epsilon$ where ϵ is a zero mean Gaussian random variable with variance σ^2. Given a labeled data set $D_i = \{(x_1, t_1), ..., (x_N, t_N) \,:\, (x_i, t_i) \sim p(x,t)\}$, we choose an estimate of t for each input x using the estimator $\hat{f}$. The estimate $\hat{f}(x)$ incurs a loss $L(t, \hat{f}(x))$. The expectation value, defined in (A.1), of the squared loss $L(t, \hat{f}(x)) = (\hat{f}(x) - t)^2$ is given by

$$\mathbb{E}[L] = \iint \left(\hat{f}(x) - t\right)^2 p(x,t)\, dx\, dt. \tag{2.1}$$

The machine learning model which achieves the minimum of the expectation value of the squared loss function is called the optimal estimator. In order to find it, we choose an $\hat{f}(x)$ which minimizes $\mathbb{E}[L]$ therefore we can apply the calculus of variations to find the minimum by setting the derivative to zero,

$$\frac{\delta \mathbb{E}[L]}{\delta \hat{f}(x)} = 2 \int (\hat{f}(x) - t) p(x,t)\, dt = 0. \tag{2.2}$$

By solving for $\hat{f}(x)$ by using the sum (A.3) and product (A.4) rules of probability,

$$\begin{aligned}
\frac{\delta \mathbb{E}[L]}{\delta \hat{f}(x)} &= 2 \int (\hat{f}(x) - t) p(x,t)\, dt = 0 \\
&= \int \hat{f}(x) p(x,t)\, dt - \int t p(x,t)\, dt \\
&= \hat{f}(x) \int p(t|x) p(x)\, dt - \int t p(x,t)\, dt \\
&= \hat{f}(x) p(x) - \int t p(x,t)\, dt \\
\implies \hat{f}(x) &= \frac{\int t p(x,t)\, dt}{p(x)} \\
&= \int t p(t|x)\, dt = \mathbb{E}[t|x],
\end{aligned} \tag{2.3}$$

the optimal estimator is recovered, namely $\mathbb{E}[t|x]$. (Note that $\mathbb{E}[t|x]$ implies $\mathbb{E}_T[t|x]$ an expectation value over a random variable T for the target; however, the T is left out for lighter notation.) Armed with the knowledge of the optimal estimator, $\mathbb{E}[t|x]$, we can expand the squared loss into an instructive form,

$$\begin{aligned}
L = (\hat{f} - t)^2 &= (\hat{f} - \mathbb{E}[t|x] + \mathbb{E}[t|x] - t)^2 \\
&= (\hat{f} - \mathbb{E}[t|x])^2 + 2(y(x) - \mathbb{E}[t|x])(\mathbb{E}[t|x] - t) + (\mathbb{E}[t|x] - t)^2
\end{aligned} \tag{2.4}$$

By substituting this expansion into the expected squared loss, the cross terms become zero and we recover

$$\mathbb{E}(L) = \int \left(\hat{f} - \mathbb{E}[t|x]\right)^2 p(x)\, dx + \int (\mathbb{E}[t|x] - t)^2 p(x)\, dx. \tag{2.5}$$

Notice that we decomposed the squared loss into the first term which measures the difference between an arbitrary estimator $\hat{f}$ and the optimal estimator, $\mathbb{E}[t|x]$, and the second term which is aleatoric and can be regarded as irreducible noise. This can be seen because the arbitrary estimator to be optimized does not enter the second term. In the next paragraphs, $h(x) \coloneqq \mathbb{E}[t|x]$ for clarity.

We turn our attention to the first term in (2.5), the difference between an arbitrary estimator and the optimal one. $\hat{f}$ models $h(x)$ using a parameter vector w, i.e. $\hat{f}(x, w)$, which is estimated by using a learning algorithm on a data set D. Since the algorithm will determine a different w depending on the data, the performance of $\hat{f}$ is evaluated by calculating the average of $(\hat{f}(x; D) - h(x))^2$ across an ensemble of data sets drawn from $p(t, x)$.

Given this insight, we add and subtract the expectation value of an arbitrary estimator across data, $\mathbb{E}_D[\hat{f}(x; D)]$ to obtain

$$\begin{aligned} &\left(\hat{f}(x; D) - \mathbb{E}_D[\hat{f}(x; D)] + \mathbb{E}_D[\hat{f}(x; D)] - h(x)\right)^2 \\ &= \left(\hat{f}(x; D) - \mathbb{E}_D[\hat{f}(x; D)]\right)^2 + \left(\mathbb{E}_D[\hat{f}(x; D)] - h(x)\right)^2 \\ &+ 2\left(\hat{f}(x; D) - \mathbb{E}_D[\hat{f}(x; D)]\right)\left(\mathbb{E}_D[\hat{f}(x; D)] - h(x)\right). \end{aligned} \tag{2.6}$$

Then we perform the analysis by taking the expectation value of this expression with respect to D. Notice that the cross term will vanish due to the linearity of expectation.

$$\begin{aligned} &\mathbb{E}_D \left[\left(\hat{f}(x; D) - h(x)\right)^2\right] \\ &= \left(\mathbb{E}_D[\hat{f}(x; D)] - h(x)\right)^2 + \mathbb{E}_D\left[\left(\hat{f}(x; D) - \mathbb{E}_D[\hat{f}(x; D)]\right)^2\right] \\ &= (\text{bias})^2 + (\text{variance}) \end{aligned} \tag{2.7}$$

As is explained succinctly in Bishop [21], "The difference between an arbitrary estimator and an optimal one is therefore decomposed into the sum of two terms. The first term, called the squared bias, represents the extent to which the average prediction over all data sets differs from the desired regression function. The second term, called the variance, measures the extent to which the solutions for individual data sets vary around their average, and hence this measures the extent to which the function $\hat{f}(x; D)$ is sensitive to the particular choice of data set."

Looking back, we found that we can express the expectation value of the squared loss 2.1 as the sum of three terms,

$$\text{expected loss} = (\text{bias})^2 + (\text{variance}) + (\text{noise}), \tag{2.8}$$

with

$$(\text{bias})^2 = \int \left(\mathbb{E}_D[\hat{f}(x; D)] - h(x)\right)^2 p(x)\, dx \tag{2.9}$$

$$(\text{variance}) = \int \mathbb{E}_D\left[\left(\hat{f}(x; D) - \mathbb{E}_D[\hat{f}(x; D)]\right)^2\right] p(x)\, dx \tag{2.10}$$

$$(\text{noise}) = \int (h(x) - t)^2 p(x, t)\, dx\, dt \tag{2.11}$$

after integrating the bias and variance. This analysis was done across many data sets but in practice we usually only have one training data set and one test data set to work with. Instead of varying the data set, a practitioner can usually only tune the model complexity. In a classical statistical learning setting this often corresponds to the number of parameters.

Models with higher complexity trained on different data sets have a better capacity to fit their training data exactly. This corresponds to an increase in variance (2.10) since the expected value of a complex model across data sets is fixed while the predictions of a specific complex model depends heavily on the training data. At the same time, when considering the expectation across all possible data sets a complex model has the capacity to capture the underlying distribution of the target and thus the bias (2.9) is reduced. This property is exactly the bias-variance trade-off and points out the reason why machine learning is difficult. Since practitioners do not have access to infinite data one must carefully tune model complexity such that the bias and variance are simultaneously minimized. For further information about this topic consider Section III in [10].

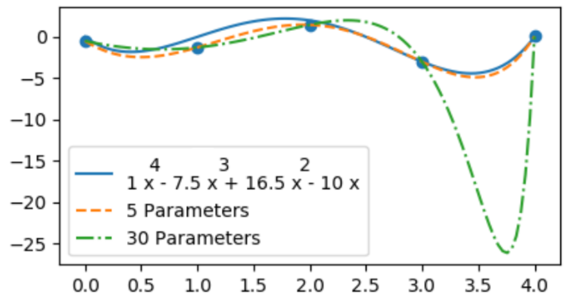

Figure 2.3: When fitting a data set (blue points) generated by a polynomial (blue) plus noise, a low complexity parameterization yields models with low variance (orange) and higher complexity yields higher variance (green).

As an example to show the increase in variance corresponding to an increase in model complexity, let us consider a polynomial fit. In this case, our target function looks like,

$$t = f(x) + \epsilon = x^4 + 7.5x^3 + 16.5x^2 + 10x + \epsilon, \tag{2.12}$$

where ϵ corresponds to Gaussian noise with mean zero and unit variance. After sampling five points from the input and target random variables, two polynomial models of differing complexity were trained on the input data. One model had five weight parameters and the other had 30. The results corresponding fit functions along with $f(x)$ and the data points are plotted in Figure 2.3. Notice that the model with 30 parameters was able to very accurately fit the training data but failed to capture the ground truth and would perform very badly if asked to do regression on new input data, for example $x = 3.5$.

This polynomial fit is empirical evidence of the bias-variance trade-off and suggests that the best generalizing model is of intermediate complexity. This concept is captured in in Figure 2.4. The plot introduces $\mathrm{E}_{\mathrm{out}}$ which measures a model's error on data drawn from $p(x, t)$ but not included in the training data set. Notice that $\mathrm{E}_{\mathrm{out}}$ is minimized at intermediate model complexity.

2.1.2 Training and Optimization of the Loss Function

The previous section introduced the expected squared loss function (2.1) in order to quantify the difference between predictions and the ground truth. Even though the optimal estimator $\mathbb{E}[t|x]$ was uncovered using the calculus of variations, it is usually impossible to solve for its exact closed formulation, especially, like in most machine learning problems, when the distribution of the data is unknown.

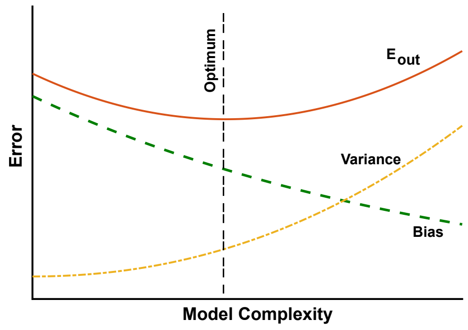

Figure 2.4: This schematic shows the typical out-of-sample error E_{out} as function of the model complexity for a training data set of fixed size. Notice how the bias always decreases with model complexity, but the variance, i.e. fluctuation in performance due to finite size sampling effects, increases with model complexity. Thus, optimal performance is achieved at intermediate levels of model complexity. The figure and caption is borrowed from [10].

Generally, we have access to the parameters w of a machine learning model $\hat{f}$ and a loss function $L(\hat{f}(x, w), t)$. The predictions of $\hat{f}$ can be improved by modifying the weights such that the L is minimized. The method explored here is called gradient descent where w is iteratively updated in the direction of large negative gradient of L. This process brings the model closer to a local minimum in L.

In most modern deep learning problems, performing gradient descent across the entire batch of training data is not computationally sensible since there are simply too many data points to iterate over. Instead, we use a modification of gradient descent applied on mini-batches of data sequentially known as stochastic gradient descent. In this case, the cost function is calculated as a sum over n data points $L = \sum_n L_n$ followed by an update step of the form

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla L_n \tag{2.13}$$

where τ is the iteration number and η is the learning rate parameter. The choice of η has a qualitative effect on descent behavior. Consider, if $\eta < 0$ then the algorithm will take steps towards higher values of L and would be better described as an ascent algorithm. By contrast, if η is too small then the descent will not reach any minima in a reasonable number of iterations [27]. A schematic of these regimes can be seen in Figure 2.5. In that picture η_{opt} implies the learning rate which will jump to the minimum in one step in a quadratic potential. In particular $\eta_{opt} = (\partial_w^2 L(w))^{-1}$.

It's worth mentioning that there exist many schemes to improve the convergence of stochastic gradient descent. A very popular one used in our experiments is called Adam [29]. Furthermore, techniques which adjust η during training have shown to reduce generalization error empirically. Due to the extremely wide availability of information about how these "tricks" work and the staggering number of possibilities out there, the author defers explanation to the search engine of your choosing.

2.2 The QM9 Data Set

The data set at the center of this work is a collection of computed geometric, energetic, electronic, and thermodynamic properties for 134k stable small organic molecules made up of CHONF [11]. This collection corresponds to the subset of all 133,885 species with up to nine heavy atoms (CONF) out of the GDB-17 chemical universe of 166 billion organic

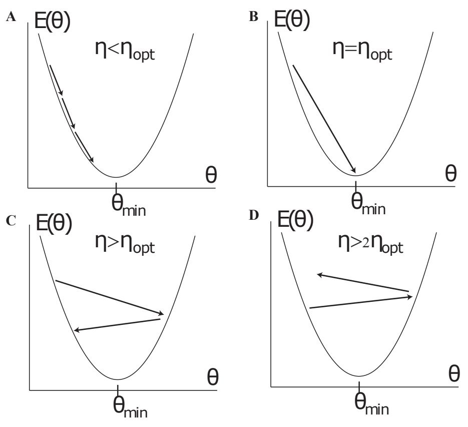

Figure 2.5: Effect of learning rate on convergence. For a one dimensional quadratic potential, one can show that there exists four different qualitative behaviors for gradient descent (GD) as a function of the learning rate η depending on the relationship between η and $\eta_{opt} = (\partial_w^2 L(w))^{-1}$. (a) For $\eta < \eta_{opt}$, GD converges to the minimum. (b) For $\eta = \eta_{opt}$, GD converges in a single step. (c) For $\eta_{opt} < \eta < 2\eta_{opt}$, GD oscillates around the minima and eventually converges. (d) For $\eta > 2\eta_{opt}$, GD moves away from the minima. This figure is drawn from [10, 28].

molecules [12]. In the field, it is known as QM9 and it has been heavily investigated, especially with data-driven methods which operate on graphs and point geometries [18, 16, 20, 30]. For our purposes, every molecule in QM9 has contains three pieces of data: Positions of constituent atoms r_a, atomic number of constituent atoms Z_a, and the set of calculated properties t_p. Table 2.2 shows the list of calculated properties relevant to this work along with converted units.

Property	Unit	Description
μ	D	Dipole moment
α	a_0^3	Isotropic polarizability
ϵ_{HOMO}	eV	Energy of HOMO
ϵ_{LUMO}	eV	Energy of LUMO
ϵ_{gap}	eV	Gap ($\epsilon_{LUMO} - \epsilon_{HOMO}$)
$\langle R^2 \rangle$	a_0^2	Electronic spatial extent
zpve	eV	Zero point vibrational energy
U_0	eV	Internal energy at 0 K
U	eV	Internal energy at 298.15 K
H	eV	Enthalpy at 298.15 K
G	eV	Free energy at 298.15 K
C_v	$\frac{\text{cal}}{\text{molK}}$	Heat capacity at 298.15 K

Table 2.1: Calculated properties in QM9 along with units and description. In the original paper, energy was presented in Hartree. a_0 is defined to be the bohr radius.

The reported property values were calculated using density functional theory which takes in $\{r_a, Z_a\}$ to estimate the electron density, the quantum mechanical energy, and other properties. The reason QM9 is a significant work is because of this problem: Imagine you only know Z_a and the bond connectivity, as is the case before quantum mechanical calculations. That means the positions r_a are unknown (along with the properties t_p). Finding the set of r_a vectors which minimizes the energy prediction of density functional theory requires an iterative optimization scheme. QM9's reported positions are the result of this optimization where the energy was computed using the B3LYP/6-31G(2df,p) level of quantum theory for most molecules and using the G4MP2 level of quantum theory for the 6095 constitutional isomers of $C_7H_{10}O_2$, the most abundant constitutional isomer. G4MP2 is a more accurate coupled cluster method, which is to say a post-Hartree-Fock method [31]. B3LYP indicates the choice of density functional while 6-31G(2df,p) indicates the choice of basis set.

Density functional theory has its roots in quantum mechanics. In particular, it is closely related to the classical Hartree-Fock approximation. Since this data set is central to the thesis, it is important to understand the basics of how the data was created, even if we do not consider further quantum mechanical calculation in this work.

2.2.1 Quantum Mechanical Calculations

The goal of quantum chemical calculation is to find the approximate solution of the time-independent, non-relativistic Schrödinger equation

$$\mathcal{H}\Psi(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N, \mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_M) = E\Psi(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N, \mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_M) \tag{2.14}$$

where $\mathcal{H}$ is the Hamilton operator for a molecule built from M nuclei and N electrons. The wave function Ψ contains all knowable information about the quantum system and E is the value of the energy of Ψ. One assumes there is no influence from external magnetic or electric fields and that, since the nuclei are >1800 times the mass of the electron, they are fixed compared to the electrons. This form of $\mathcal{H}$ is called the Born-Oppenheimer approximation. The electron contribution looks like,

$$\mathcal{H}_e = -\frac{1}{2}\sum_{i=1}^{N} \nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M} \frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{1}{r_{ij}}, \tag{2.15}$$

where ∇_i^2 is the Laplacian on the electrons, Z_A is the atomic number of atom A, r_{iA} is the distance between electron i and nucleus A, and r_{ij} is the distance between electrons i and j. The terms encode the kinetic energy of the electrons, the interactions between the electrons and the nuclei, and the interaction between electrons. To accurately calculate the energy, one must reintroduce the constant nuclear repulsion term,

$$E_n = \sum_{A=1}^{M}\sum_{B>A}^{M} \frac{Z_A Z_B}{r_{AB}}, \tag{2.16}$$

resulting in, $E = E_e + E_n$. Where $\mathcal{H}_e\Psi = E_e\Psi$. The problem is that, in general, finding the wavefunction which fulfills this eigenvalue equation is not possible analytically, so we turn to approximation. The variational principle saves the day, recall that

$$\langle \Psi_{trial}|\mathcal{H}|\Psi_{trial}\rangle = E_{trial} \geq E_0 = \langle \Psi_0|\mathcal{H}|\Psi_0\rangle \tag{2.17}$$

where Ψ_0 is the true energy of the ground state. In other words, the lower bound on the expectation value of the Hamiltonian is the true ground state energy and the true ground state energy is recovered if and only if the trial wavefunction is identical to the true wavefunction. This is useful because it allows us to solve the problem by minimizing the expression $\Psi_0 = \arg\min_\Psi \langle \Psi|\mathcal{H}|\Psi\rangle$.

In practice, one uses Hatree-Fock approximation to find the minimum from a suitable subset of possible wavefunctions. It consists of approximating the N-electron wave function by an antisymmetrized product of N one-electron wave functions $\chi_i(\mathbf{x}_i)$ known as spin-orbitals. The product is called a Slater determinant,

$$\Psi \approx \Phi_{SD} = \frac{1}{\sqrt{N!}} \det(\chi_i(\mathbf{x}_j)) \;\; i,j = 1,2,...,N. \tag{2.18}$$

Each of these spin-orbitals solves an eigenvalue equation involving the Fock operator,

$$f = -\frac{1}{2}\nabla_i^2 - \sum_A^M \frac{Z_A}{r_{iA}} + V_{HF}. \tag{2.19}$$

The first two terms are the kinetic energy and potential energy due to interaction with the nucleus. The final potential element is the mean-field repulsive force experienced by an electron due to the other electrons. It consists of a Coulomb interaction between an electron and the charge density of the other electrons in the other spin orbitals as well as a purely non-classical term called the exchange operator. Since electrons are fermions, the wavefunction must be anti-symmetric when one swaps same-spin electrons. The exchange operator implements this criterion. Since the fock operator depends on the solution to the mean field, V_{HF}, the eigenstates must be found iteratively. After an initial guess, the iterative approach is taken until numerical convergence, known as self consistency.

The accurate application of Hartree-Fock depends on the judicious choice of basis set. One often applies a linear combination of atomic orbitals. An example set of atomic orbitals are the Slater-type orbitals which have exponential radial decay, satisfy Kato's cusp condition (for more information see [32]), and have no nodal structure (unlike the hydrogen wave function). Another example which approximates the Slater-type orbitals are the Gaussian-type orbitals which decay faster with a square exponential radial decay, do not satisfy the cusp condition, and also have no nodal structure. The reason they are so often used is because of the Gaussian Product Theorem. It constrains the product of two Gaussians to be another Gaussian, avoiding integration and decreasing calculation time by orders of magnitude.

The basis set 6-31G(2df,p) used in calculating the electron densities for most of QM9 is a set of Gaussian-type orbitals. It is a so-called split-valence basis set where core orbitals are modeled using a single basis function while valance orbitals are modeled by more than one. In order to improve the quality of the atomic basis set, a fixed weighted sum of six Gaussian-type orbitals, called a contracted Gaussian-type orbital, is sometimes used. In the case of 6-31G(2df,p), the core orbitals are a contraction of six Gaussian-type orbitals while the basis set for valance orbitals contains both a contraction of 3 Gaussian-type orbitals and another uncontracted Gaussian-type orbital. The remaining letters indicate inclusion of polarization and asymmetries into the basis sets by modulating the Gaussian-type orbitals with the spherical harmonics, see Section 2.3.3. Specifically, two sets of d functions and one set of f functions are added to heavy atoms (CONF) and one set of p functions are added to hydrogen. Recall that p implies spherical harmonics of order 1, d implies order 2, and f implies order 3.

The majority of compounds were calculated using density functional theory. The Hohenberg-Kohn Theorems prove that for any potential, up to a constant factor, there exists a unique ground state electron density. This has the consequence that knowing the electronic density is enough to completely characterize all knowable properties of a molecular system

[32]. Density functional theory studies the functionals which map from density to properties, such as ground state energy. Although it is possible to write them down symbolically, only some of their explicit formulation is known and other parts must be approximated. In particular, given a ground state electron density ρ_0 and three functionals yielding kinetic energy, electron-electron interaction energy, and electron-nucleus interaction energy T, E_{ee}, and E_{Ne}, we write the ground state energy as

$$E_0[\rho_0] = T[\rho_0] + E_{ee}[\rho_0] + E_{Ne}[\rho_0]. \tag{2.20}$$

The nuclear-electron interaction term is defined $E_{Ne}[\rho] = \int p(\mathbf{r}) V_{Ne} d\mathbf{r}$ with V_{Ne} defining the potential seen by the electrons from the nuclei. The electron-electron interaction is expanded to,

$$E_{ee}[\rho_0] = \frac{1}{2} \iint \frac{\rho_0(\mathbf{r}_1)\rho_0(\mathbf{r}_2)}{r_{12}} d\mathbf{r}_1 d\mathbf{r}_2 + E_{qm}[\rho_0], \tag{2.21}$$

which includes a Coulomb interaction term and a term which contains all quantum electronic contributions including self-interaction correction, exchange of particles, and Coulomb correlation. That leaves two functionals which are unknown explicitly, E_{qm} and T. Approximating these functionals has the effect of perturbing the Hamiltonian which implies that the variational principle no longer holds, i.e. it is possible to return an energy which is lower than the true ground state energy. This high price confers the benefit that it is significantly cheaper to compute properties using density functionals than with Hartree-Fock or related methods.

The successful approximation of these functionals is called the Kohn-Sham approach, whereby as much of the kinetic energy is computed exactly as possible while leaving the remaining portion for approximation. The kinetic energy of a non-interacting reference system T_S with an identical ground state electron density can be computed exactly using a spin-orbital basis, as above. Given this decomposition, the E_{qm} functional is combined with a functional for the unaccounted kinetic energy $T_C = T - T_S$ and named the exchange correlation functional, $E_{XC} = T_C + E_{qm}$. The choice of functional plays an important role in the accuracy of results. B3LYP (Becke three-parameter Lee Yang Paar) is a so-called hybrid functional which is a linear combination of the Hartree-Fock exchange functional and other ad-hoc density functionals. The linear combination is set empirically.

2.3 Geometric Quantities

Consider a gas of atoms represented as a point cloud each with their own velocities and accelerations. When that point cloud is rotated, the associated velocities and accelerations rotate along with it. Each of those vectors is a geometric quantity and their rotation properties are what differentiates them from an arbitrary grouping of components. When presented after the atomic gas example, this may seem obvious; however, the distinction becomes clearer when comparing a geometric vector to other arrays commonly used in machine learning. Imagine instead a single frame from a movie of individually colored ping pong balls represented as points, each with their own colors encoded as an array of RGB scalars. When the scene is rotated, the ping pong balls rotate but since the colors do not change upon rotation, the components of the RGB color array do not change. Unlike velocity and acceleration, the RGB color array is a non-geometric arbitrary grouping of scalars.

The design principle in e3nn, the core neural network in this study, is to utilize features which are geometric quantities and therefore have well-defined rotation properties. The effect of this choice is that two copies of the same input data presented in different orientations would produce the same features in the network up to a rotation. For molecular systems, this property is invaluable because they do not have a canonical orientation.

In order to implement this, it is useful to think of all possible rotations and translations which ought to be well-defined on the features. This collection of transformations is called a group. Those transformations act on the features by means of a representation. In this work, we are interested in the group of rotations and translations. This section formalizes how e3nn's features transform.

2.3.1 Groups and Representations

The mathematical term for a set coupled with an operation that maps between elements of the set is called a group. Important groups have names. For our purposes, an interesting group is the special orthogonal group, SO(3): The set of rotations on $\mathbb{R}^3$ paired with composition. The set of rotations and translations on $\mathbb{R}^3$ paired with composition is called the special euclidean group, SE(3). When the parity operation is included as well, the group is called the euclidean group, E(3). For completeness, we define a group.

Definition 2.3.1 *A Group, G, is a set with a rule for assigning to every (ordered) pair of elements, a third element, satisfying the following axioms:*

Closure If $f, g \in G$ then $h = fg \in G$.

Associativity For $f, g, h \in G, f(gh) = (fg)h$.

Identity There is an identity element, e, such that for all $f \in G, ef = fe = f$.

Inverse Every element $f \in G$ has an inverse, f^{-1}, such that $ff^{-1} = f^{-1}f = e$.

When the underlying set of a group happens to be a smooth manifold and the group action is compatible, that group is called a Lie group. The action is compatible when the operators $\times : G \times G \to G$, and $(\cdot)^{-1} : G \to G$ are differentiable. Both SO(3) and SE(3) are examples of Lie groups.

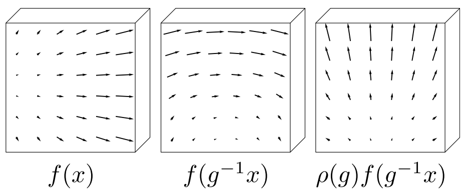

Figure 2.6: To transform a vector field (left) by a 90° rotation g, first move each arrow to its new position (center), keeping its orientation the same, then rotate the vector itself (right). This is described by the representation $\pi = \rho$, where $\rho(g)$ is a 3×3 rotation matrix that mixes the three coordinate channels. Figure and caption adapted from [33].

Let us consider the action of translation and rotation on a three-dimensional vector field defined by the map $f : \mathbb{R}^3 \to \mathbb{R}^3$, i.e. every point $x \in \mathbb{R}^3$ in space is associated with a vector. Translation by t is simple, move each vector from the initial position x to $x-t$, i.e. $f(x) \mapsto f(x-t)$. When rotated by r, first the vector at $r^{-1}x$ is moved to a new position x and each vector is also rotated by a rotation matrix $\rho(r) \in \mathbb{R}^{3\times3}$, i.e. $f(x) \mapsto \rho(r)f(r^{-1}x)$. The rotation matrix ρ introduces a coupling or dependency between the different channels of $f(x)$. ρ is what makes this vector quantity geometric; just as stated in the chapter's introduction.

Since we are interested in translations and rotations together, we consider the decomposition of $g \in \mathrm{SE}(3)$ into a rotation $r \in \mathrm{SO}(3)$ and translation $t \in \mathbb{R}^3$ written as $g = tr$. The transformation law for the vector field f is given by

$$[\pi(tr)f](x) \coloneqq \rho(r)f(r^{-1}(x-t)). \tag{2.22}$$

Where π is known as the representation of SE(3) induced by the representation ρ of SO(3). (While representations are covered next, induced representations are beyond the scope of this work, see [33].)

2.3.2 Group Representation

An n dimensional representation ρ maps every element in the group to an an invertible $n \times n$ matrix. As seen in the last section, the form of this representation is the defining geometric property of the field f. A representation is defined below [34].

Definition 2.3.2 *A Representation of G is a mapping, D, of the elements of G onto a set of linear operators with the following properties:*

1. *$D(e) = 1$ where 1 is the identity operator in the space on which the linear operators act.*
2. *$D(g_i)D(g_j) = D(g_i g_j)\ \forall g_i, g_j \in G$, in other words the group multiplication law is mapped onto the natural multiplication in the linear space on which the linear operators act.*

In SO(3), every representation can be created from building blocks of so-called irreducible representations. This concept is rather abstract so we consider an example from [33] about a rank-2 tensor (i.e. a matrix). Our 3×3 matrix A transforms under rotation like $A \mapsto R(r)AR(r)^T$, where $R(r)$ is the 3×3 rotation matrix representation of the abstract group element $r \in \mathrm{SO}(3)$. By flattening the matrix A into a vector using $vec : \mathbb{R}^{3\times 3} \to \mathbb{R}^9$, we can rewrite the transformation using the tensor product form as $vec(A) \mapsto [R(r) \otimes R(r)]\, vec(A) \coloneqq \rho(r)vec(A)$. This example is a 9-dimensional representation of SO(3).

A can be divided up into two different linear subspaces, namely the symmetric and anti-symmetric parts. The reason is because the 6-dimensional symmetric linear subspace of A remains symmetric under rotations. The same goes for the 3-dimensional anti-symmetric part. Since these two subspaces transform independently under rotation, they can be considered distinct even if this is not obvious by analysis of the coordinates of A_{ij}. The 6-dimensional symmetric subspace can be further broken down because scalar matrices $A_{ij} = \alpha\delta_{ij}$ are invariant under rotation and transform independently from traceless symmetric matrices. We are left with three independently transforming subspaces of dimension 1 (trace), 3 (anti-symmetric part), and 5 (traceless symmetric part). This division is called reducing the 9-dimensional representation ρ into irreducible representations of dimension 1, 3, and 5. Since this decomposition is possible, it implies that there exists a change of basis matrix Q such that,

$$\rho(r) = Q^{-1}\left[\bigoplus_{l=0}^{2} D^l(r)\right]Q, \tag{2.23}$$

where $\oplus$ implies the construction of a block-diagonal matrix with blocks $D^l(r)$. This happens to hold in general, which means that any representation of SO(3) can be decomposed into a direct sum, $\oplus$, of irreducible representations of dimension $2l+1$ for $l = 0, 1, 2, ...$

It is natural to discuss representations of SO(3) in terms of direct sums of irreducible representations. Specifically, each irreducible linear operator representation of SO(3) is known as a Wigner D-Matrix, $D^l(r)$. These matrices are linear operators on the vector space of spherical harmonics Y^l_m.

2.3.3 The Spherical Harmonics and the Wigner D-Matrices

The spherical harmonics are a complete orthonormal basis for functions on the sphere. They arise from many different settings including the solution of Laplace's equation on the sphere, in the modeling of electric and magnetic dipoles, and they can be seen as angular frequencies for the Fourier transformation on a sphere. For our purposes, they serve as the basis for a vector space paired with the orthogonal linear operators on that vector space, namely the Wigner D-Matrices.

In e3nn, every feature is encoded as a spherical harmonic, computed using the package Lie Learn which in turn draws on SciPy [35, 36]. The complex-valued spherical harmonics $\tilde{Y}^l_m$ are defined like so,

$$\tilde{Y}^l_m(\theta,\phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}e^{im\theta}P^l_m(cos(\phi)) \tag{2.24}$$

$$P^l_m(x) = (-1)^m(1-x^2)^{m/2}\frac{d^m}{dx^m}P^l(x) \tag{2.25}$$

$$P^l(x) = \sum_{k=0}^{\infty}\frac{(-l)_k(l+1)_k}{(k!)^2}\left(\frac{1-x}{2}\right)^k, \tag{2.26}$$

where P^l_m are the associated Legendre functions of integer order and real degree and the Condon-Shortley phase convention is used [37]. e3nn operates only on the real-valued spherical harmonics Y^l_m which can be determined from the complex valued spherical harmonics using the following transformation,

$$Y^l_m = \begin{cases} \frac{i}{\sqrt{2}}(\tilde{Y}^l_m - (-1)^m\tilde{Y}^l_{-m}) & m < 0 \\ Y^0_l & m = 0 \\ \frac{1}{\sqrt{2}}(\tilde{Y}^l_{-m} + (-1)^m\tilde{Y}^l_m) & m > 0. \end{cases} \tag{2.27}$$

Since the transformation is unitary, the orthogonality and normalization properties of Y^l_m are the same as those of the $\tilde{Y}^l_m$. An image of the first few orders of spherical harmonics using these conventions are shown in Figure 2.7

How do the spherical harmonics rotate, i.e. what are the irreducible representations of SO(3)? First note that the pair (θ, ϕ) can be written as a unit vector $\hat{x}$. (The hat is not to be confused with a prediction like in most other sections of this document.) With this notation, the spherical harmonics are written $Y^l_m(\hat{x})$ and given rotation $r \in$ SO(3) rotate like,

$$Y^l_m(\hat{x}) \mapsto Y^l_m(r^{-1}\hat{x}) = D^l(r)Y^l_m(\hat{x}), \tag{2.28}$$

where $D^l(r)$ is a Wigner D-Matrix. Computing this quantity becomes involved rather quickly so we suffice to say that given a rotation $r(\alpha, \beta, \gamma)$ parameterized by the z-y-z Euler angles [38], the Wigner D-Matrix is defined using the inner product,

$$D^l_{m'\,m} = \langle Y^l_{m'} | r(\alpha, \beta, \gamma) | Y^l_m \rangle. \tag{2.29}$$

For further conventions consult the Lie Learn package [35] or other standard sources about representation theory.

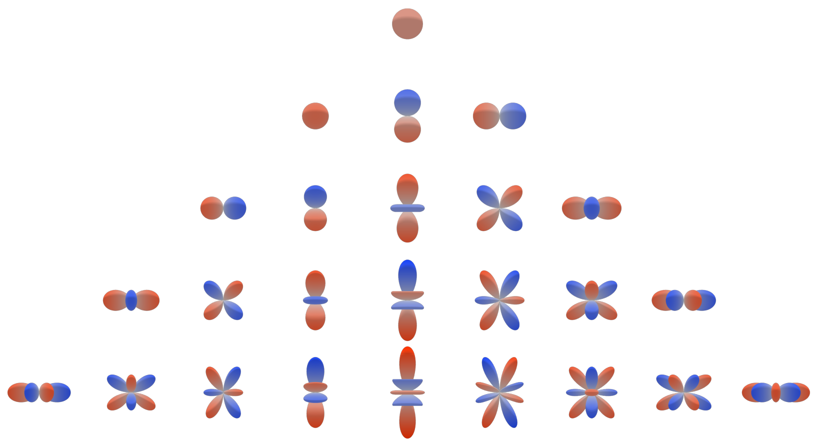

Figure 2.7: These are the real spherical harmonics in e3nn's convention. The order l increases from top to bottom starting with zero and ending with four. Each of the columns corresponds with setting the component with index m equal to one, while the other indices are set to zero. Notice that the reason there are more spherical harmonics at higher orders is because the number of m indices is $|m| = 2l + 1$.

2.3.4 The Clebsch-Gordan Decomposition

Pairs of spherical harmonics interact with each other via the tensor product in e3nn like $Y^{l_1}_m \otimes Y^{l_2}_{m'}$. The output transforms with the tensor product of the corresponding irreducible representations,

$$Y^{l_1}_m \otimes Y^{l_2}_{m'} \mapsto (D^{l_1}(r) \otimes D^{l_2}(r))(Y^{l_1}_m \otimes Y^{l_2}_{m'}).$$

In general $(D^{l_1}(r) \otimes D^{l_2}(r))$ is not an irreducible representation of SO(3); however, it does have a decomposition into irreducibles called the Clebsch-Gordan decomposition [18]:

$$D^{l_1}(r) \otimes D^{l_2}(r) = C^{-1}_{l_1,l_2} \left[\bigoplus_{l=|l_1-l_2|}^{l_1+l_2} D^l(r) \right] C_{l_1,l_2} \tag{2.30}$$

where C_{l_1,l_2} is a change of basis matrix called the Clebsch-Gordan coefficients. This matrix is normally presented as $C_{l_1,l_2,l} \in \mathbb{R}^{(2l+1)\times(2l_1+1)(2l_2+1)}$ which is formed by taking the block of $2l + 1$ rows from C_{l_1,l_2} corresponding to the l component in the direct sum. Note that the product $C_{l_1,l_2,l}(Y^{l_1}_m \otimes Y^{l_2}_{m'})$ is an lth order spherical harmonic.

At this point, all of the necessary rotation properties have been defined. We know how scalars, vectors, spherical harmonics, and tensor products of spherical harmonics rotate. (For clarity, scalar s rotates with arbitrary rotation r like $s \mapsto s$.) With this knowledge, it is possible to define a neural network which has inputs and outputs with well-defined rotation properties.

2.4 Neural Networks Equivariant to SE(3)

Neural networks are often more successful when their design takes the geometry of the underlying data into account. The most widespread example is the convolutional neural network which introduces a strong prior that locality is necessary for determining useful features. In contrast to multi-layer perceptrons, convolutional neural networks do not allow arbitrary connections but rather calculate their features by passing a learned convolutional kernel over the data.

We say that the parameters of a convolutional neural network are shared because every part of the input data is treated using the same convolutional kernel. This reduces the number of learned parameters and thus the model size. In addition, it introduces equivariance to translation, i.e. translated input features return a translated output feature. In this way, the network can easily learn to find features common throughout the input space. For a visual example see Figure 2.8.

When a function f commutes with the action of a group $g \in G$, the function is said to be equivariant to that group, $f \circ g = g \circ f$. The natural way to enforce the prior that local features can be found anywhere in the input field is through translation equivariance. For our problem, we want to enforce the prior that molecular features can be found at arbitrary translations, rotations, and with arbitrary permutations of atoms. We achieve this through SE(3) equivariance.

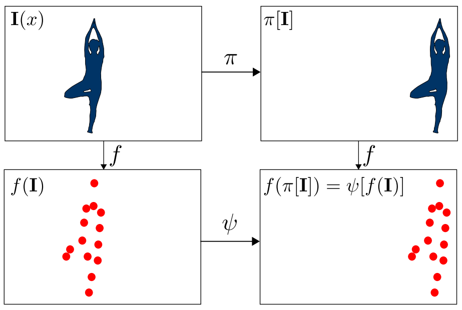

Figure 2.8: Let f be a function which transforms the dancer into a dot representation. Let π be translation the domain of f while ψ is translation in the codomain of f. The function f is said to be equivariant to translation when the right-down path or the down-right path yields the same result. A convolution neural network is equivariant to translation up to the convolution kernel. Image from [39].

The SE(3) equivariant neural network presented in this section, e3nn, takes the geometry of molecules represented as point clouds into account by utilizing continuous, atom-centered convolutional kernels. Translation equivariance is achieved by only considering the difference of atomic positions. Rotation equivariance is achieved by incorporating only rotationally invariant operations like multiplication by a scalar and the rotationally equivariant Clebsch-Gordan product. In addition, parameters are shared across convolutional kernels. Since each atom takes its turn acting as the convolutional center with shared convolutional kernel parameters, this has the effect of treating the atoms like a set and therefore introducing equivariance to permutation of atoms.

2.4.1 e3nn

The E(3) equivariant neural network utilized in this work takes labeled point geometry and associates learned features to every point, represented as spherical harmonics, which are correlated with the arrangement of the geometry itself. These features can be used for regression on labeled data as in Section 3.3, as displacement vectors, and as higher-order spherical Fourier representations of the local environment.

The development of the E(3) equivariant neural network was intertwined with the necessities and results of wide-ranging experiments beyond what is presented in this work. Initial experiments utilized the Tensor Field Network [40] implementation. The regression experiments in Section 3.3 called for more computational efficiency. These concerns were met using e3nn [17], the result of the cross pollination between Tensor Field Networks and se3cnn [33], an SE(3) equivariant neural network restricted to 3d volumetric data.

Features and Geometry

The core `torch.tensor` objects on which e3nn operates are called `feature`, which stores the irreducible representations of the spherical harmonic signal on every atom, and `geometry`, which contains the positional information of every atom. `geometry` is a rather simple multi-dimensional array of batches of atoms in $\mathbb{R}^3$.

The `feature`, on a specific atom, encodes the mathematical tensor F^l_{ui}. The index l corresponds to the order of the spherical harmonic, u identifies the multiplicity, i indexes the components of a spherical harmonic of order l which range from 0 to $2l + 1$. Due to the dependence of i on l and the option to have different multiplicities for each l, the components representing this object cannot, in general, be stored in a rectangular multi-dimensional array; however, it is possible to list them by organizing the list first by order l, then by multiplicity u, then by representation component i. (Imagine the generalization of the *vectorize* : $\mathbb{R}^{N\times N} \to \mathbb{R}^{N^2}$ operation on matrices.) `feature` stores exactly said list.

Each `feature` has a representation shorthand, `Rs`, associated with it in order to address the irreducible representations of the spherical harmonics it contains. `Rs` is a list of tuples where every tuple has the form `(multiplicity, order)`. The length of a `feature` can be determined using a list comprehension, like in the `dim` function.

```
def dim(Rs):
    return sum(mul * (2 * l + 1) for mul, l in Rs)
```

An important point to understand is that, although `feature` is flat, it contains a direct sum of irreducible representations of SO(3) which must be treated with great care to retain the symmetries of the problem. Arbitrary operations along the array, like norm, are usually not mathematically defined.

Consider an example, let F^l_{ui} be a tensor of two zeroth order spherical harmonics and one first order spherical harmonic with $l \in \{0, 1\}$, $u(l = 0) \in \{1, 2\}, u(l = 1) \in \{1\}$, $i(l) \in \{1, ..., 2l + 1\}$,

$$\begin{aligned} F^0_{11} &= a \quad & F^1_{11} &= c \\ F^0_{21} &= b \quad & F^1_{12} &= d \\ & & F^1_{13} &= e. \end{aligned} \tag{2.31}$$

Given `Rs = [(2, 0), (1, 1)]`, there are two possible arrangements of the data `feature = tensor([a, b, c, d, e])` and `feature = tensor([b, a, c, d, e])`. We could also take the same F^l_{ui} but use a different `Rs' = [(1, 0), (1, 1), (1, 0)]`, which would yield either `feature' = tensor([a, c, d, e, b])` or `feature' = tensor([b, c, d, e, a])`. We have enumerated some of the possible arrangements of `feature`; however, for the network to learn, the user must pick a convention and stick to it. The layout is such that the representation components for any specific spherical harmonic are contiguous in `feature` and the multiplicities are ordered consistently. Therefore, remember, while e3nn is permutation equivariant with respect to atoms, `feature` is **not** permutation equivariant with respect to the indices l, u, or i.

`feature`'s flat data arrangement allows for the creation of a rectangular multi-dimensional array where one dimension is for features and the other dimensions are for atoms or batches. This is useful so that `geometry` and `feature` can share the first indices, i.e.

```
>>> feature.size()
torch.size([batch, atom, dim(Rs)])
>>> geometry.size()
torch.size([batch, atom, 3])
```

With this formulation it becomes simple to address specific atoms. For example, atom 2 in batch 3 has features `feature[3, 2, :]` and is located at position `geometry[3, 2, :]`. A word of warning about the different bases in `feature` and in `geometry`. Even if `Rs = (1, 1)`, `feature` is not a vector in the same basis as `geometry`, rather it is a spherical harmonic centered on an atom and a change of basis is required in order to convert it to a displacement vector from that atomic center. The tools to do so are in the `e3nn.o3` module and more information about spherical harmonics can be found in Section 2.3.3.

How do `feature` and `geometry` transform under rotation? Consider a rotation $\mathbf{R} \in SO(3)$. `geometry` is written as an array of vectors r_a and therefore it rotates like,

$$r_a \mapsto \mathbf{R} r_a. \tag{2.32}$$

The tools for rotating geometry are available in the `e3nn.o3` module under the name `rot` which is parameterized using Euler angles.

Let F^l_{ui} be the spherical harmonic signal encoded in `feature`. The features are always written in an irreducible representation therefore we can write their rotation matrix as a direct sum of Wigner D-matrices. Therefore F^l_{ui} rotates like,

$$F^l_{ui} \mapsto \left(\bigoplus_{l'=1}^{l} \bigoplus_{u'=1}^{u} D^{l'}_{u'}(\mathbf{R}) \right) F^l_{ui}. \tag{2.33}$$

This feature is in the module `e3nn.rs` module under the name `rep` which is parameterized using Euler angles and the `Rs` for the feature.

The Neural Network Layers

Now that we have introduced the core objects in e3nn, we ought to explain the operations which are possible on those objects. This work will discuss the mathematics behind the neural network building blocks of e3nn which are utilized in the experiments of Chapter 3. The network requires a learned radial model to expand the distances between atoms in a radial basis using a continuous filter, a kernel and convolution which generates new features from the geometry and combines them with existing features, and finally a gated nonlinearity.

In the following tensor contractions and notations, note that the Einstein summation convention is **not** employed, i.e. all summed indices are listed explicitly under the summation symbol. When an index appears twice in the summand, but does not appear under the summation symbol, then it is considered to be multiplied index-wise. Note that the upper

Index	**Description**
l_{out}, l_{in}, l_f	Order of spherical harmonic. Output, input, and filter respectively.
u, v	Multiplicity.
i, j, k	Representation components of spherical harmonics at order l. Often called m in literature. Can take $2l+1$ values.
A, B	Output atom index (which atoms take the new features), Convolved atom index.
Z	Batches.
$\mathcal{B}, \mathcal{H}, \mathcal{I}, \mathcal{O}$	In the radial model, corresponds to basis, hidden, and output indices respectfully. Each ranges from 1 to $n_{basis}, n_{hidden}, n_{in}$, and n_{out}.

Table 2.2: Index reference table.

or lower positioning of an index has no meaning when it comes to co- or contravariance of that index. Since our problem is set in an orthonormal coordinate system, the transmutation of vectors to covectors is merely the transpose operation. It is assumed that one out of every pair of indices, i, j, k, is covariant in order to contract the tensor. This only matters when it comes to calculating the Clebsch-Gordan decomposition. In general, the other arrays are scalars like the radial model or the normalization factors and therefore their co- or contravariance does not matter at all. Since the next few sections rely heavily on many different indices, Table 2.2 acts as a reference to determine what is meant by each particular index.

Radial Model

A discontinuous filter, like the ones used in convolutional neural networks, do not capture subtle differences in position like continuous filters do. See Figure 2.9. The notion of using a continuous learned convolutional kernel originated in SchNet in order to predict smooth energy surfaces [16]. Like SchNet, e3nn's radial model takes the magnitude of the distance between atoms and expands it in a continuous radial basis. A non-linear combination of basis elements using a multi-layer perceptron learns to scale, atom-wise and multiplicity-wise (u, v), existing (l_{in}), convolved (l_{filter}), and output (l_{out}) features in the kernel. The scalars predicted from the radial model are how e3nn learns. Their usage is made explicit in the following section about the kernel.

The magnitude of the distance between atoms A and B is written $d_{AB} = \|\mathbf{r}_B - \mathbf{r}_A\|$. Expanding d_{AB} in an arbitrary radial basis with n_{basis} basis elements $\phi : \mathbb{R}_+ \to \mathbb{R}^{n_{basis}}$ yields an array of scalars which we can address element-wise using $D^{\mathcal{B}}_{AB} = \phi(d_{AB})$ with $\mathcal{B} = 1, .., n_{basis}$. A normal dense layer $\mathcal{L}^{\mathcal{O}} = \sigma(\sum_{\mathcal{I}} x^{\mathcal{I}} w^{\mathcal{I}}_{\mathcal{O}})$, $\mathcal{I} = 1, ..., n_{in}$, $\mathcal{O} = 1, .., n_{out}$ with arbitrary input $x \in \mathbb{R}^{n_{in}}$, learned parameter matrix $w \in \mathbb{R}^{n_{in} \times n_{out}}$, and non-linearity σ can be stacked to create a multi-layer perceptron. The radial model is flexible and can use arbitrarily deep layers with an arbitrary number of hidden units to produce a non-linear combination of distances expanded in the radial basis. The first layer of the radial model using the basis expansion is written,

$$\mathcal{L}^{\mathcal{H}}_{AB} = \sigma\left(\sum_{\mathcal{B}} D^{\mathcal{B}}_{AB} w^{\mathcal{B}}_{\mathcal{H}}\right), \mathcal{H} = 1, ..., n_{hidden} \ . \tag{2.34}$$

If we choose $n_{out} = \max l_{out} \times \max l_{in} \times \max l_f \times \max u \times \max v$, we generate enough scalars for the kernel. Therefore we can write the final layer of the linear model in terms of the indices used in the kernel and the output of the previous hidden layer $\mathcal{L}^{\mathcal{H}}$,

$$R^{l_{out} l_{in} l_f}_{uv} = \sum_{\mathcal{H}} \mathcal{L}^{\mathcal{H}} \ w^{\mathcal{H} \, l_{out} l_{in} l_f}_{uv}, \tag{2.35}$$

since, in the output layer, σ is the identity function. Note that since the A and B indices persist through every layer, including the last one, it is more general to leave them out, thereby simplifying the notation and offering the option of considering a radial model from any positive real number input, i.e. $d_{AB} = d$.

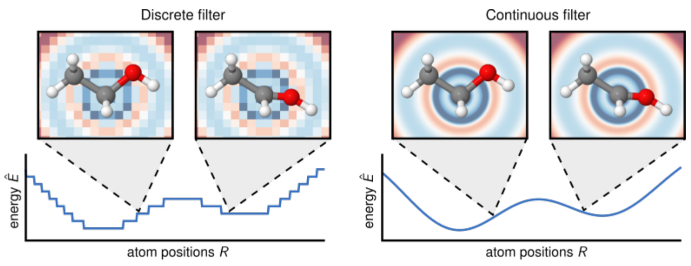

Figure 2.9: In this image from [16], one can see how using a discrete filter is not appropriate for estimating energy since the prediction is discontinuous. Instead, a basis of continuous filters allows for the prediction of a smooth energy function across all atomic positions. The colors in the background indicate the learned radial filter while the atom in the foreground is indicating an energy depending on position.

The choice of basis was shown to play an important role in the newly released DimeNet [20] which used radial Bessel functions. The bases considered in this work include the Gaussian basis ϕ_G, the cosine basis ϕ_C, and the radial Bessel basis ϕ_B. ϕ_B calls for a cutoff c where interactions $d > c$ go to zero. The Gaussian and cosine bases are centered with $\mu_{\mathcal{B}} \in \mathbb{R}_+$ as the center for each basis element. If the centered basis elements were each consistently separated by $\mu_{\mathcal{B}+1} - \mu_{\mathcal{B}}$, then $\gamma = 1/(\mu_{\mathcal{B}+1} - \mu_{\mathcal{B}})$. The bases are,

$$\phi_G = \exp(-\gamma \|d - \mu_{\mathcal{B}}\|^2) \tag{2.36}$$

$$\phi_C = \begin{cases} cos^2(\frac{\pi}{2}\gamma(d - \mu_{\mathcal{B}})) & -1 \leq \gamma(d - \mu_{\mathcal{B}}) \leq 1 \\ 0 & otherwise \end{cases} \tag{2.37}$$

$$\phi_B = \sqrt{\frac{2}{c}} \frac{sin(\frac{\mathcal{B}\pi}{c} d)}{d}. \tag{2.38}$$

Fifteen ϕ_G, ϕ_C bases with evenly spaced $\mu_{\mathcal{B}}$ are plotted in Figure 2.10 along with the first ten ϕ_B with $c = 10$.

Kernel

The kernel serves two primary purposes: First, it acts like a partial evaluation of the Clebsch-Gordan decomposition of the tensor product of two spherical harmonic signals by

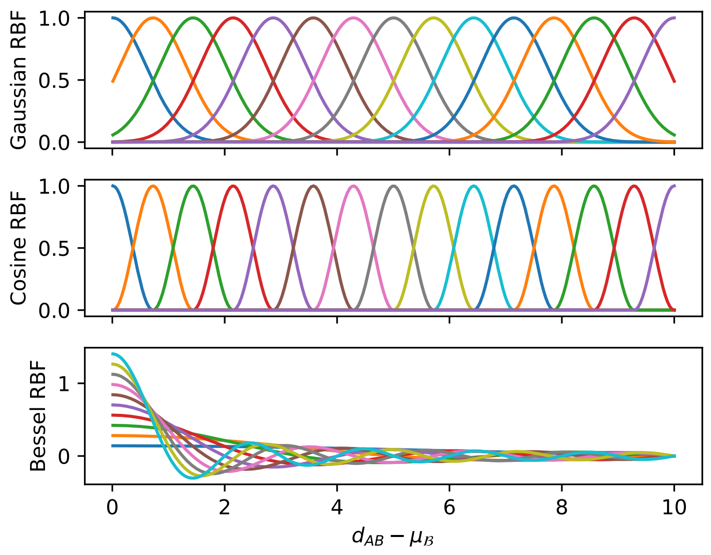

Figure 2.10: The radial bases are plotted, in different colors for each $\mathcal{B}$ basis, versus $d_{AB} - \mu_{\mathcal{B}}$. The ϕ_G is non-zero for the entire positive number line with a much slower decay while ϕ_C is zero everywhere except on the region $-1 \leq \gamma(d_{AB} - \mu_{\mathcal{B}}) \leq 1$. ϕ_B is not centered but instead higher index $\mathcal{B}$ corresponds to larger oscillations. Past the cutoff, 10 in this example, ϕ_B takes the value zero.

filling in the signal generated by the radial filter. Secondly, assuming the output of the radial model, $R_{uv}^{l_{out} l_{in} l_f}$, is normalized, the kernel output is normalized through a multi-dimensional array of scalars $n^{l_{out} l_{in}}$.

e3nn generates v multiplicity of filters with order l_f and components $k \in \{-l_f, -l_f + 1, ..., l_f - 1, l_f\}$ by evaluating the radial model on the distance between atoms d_{AB} and the spherical harmonics on the unit vector between atoms $\hat{\mathbf{r}}_{AB} = \frac{\mathbf{r}_B - \mathbf{r}_A}{d_{AB}}$. The resulting filters can be indexed using,

$$R_v^{l_f}(d_{AB}) Y_k^{l_f}(\hat{\mathbf{r}}_{AB}). \tag{2.39}$$

e3nn combines this filter with the existing spherical harmonic signal on each atom using the Clebsch-Gordan decomposition of the tensor product of two spherical harmonics, see Section 2.3.4. If we call this function G and the existing features F, we can sketch the function G (with significant omission of indices) by writing,

$$G(R(d_{AB})Y(\hat{\mathbf{r}}_{AB}), F). \tag{2.40}$$

The kernel K represents partial evaluation of G where (2.39) is already “filled in”, i.e.

$$K(\cdot) \coloneqq G(R(d_{AB})Y(\hat{\mathbf{r}}_{AB}), \cdot). \tag{2.41}$$

The advantage of partial evaluation of G is that there is more flexibility computationally. By computing K, one can reuse the filter multiple times without having to recompute it. However, in the regression task in Section 3.3, this technique significantly slowed down the calculation and increased the memory footprint. In order to combat this problem, the author defined a function which computed G directly in PyTorch. This is discussed below, in the subsection KernelConv.

The remaining piece of the kernel is the normalization, n. The goal is to have a convolution output G in which the magnitude of each spherical harmonic is normalized. We assume that the radial model R is mean zero with unit variance and the magnitude of each spherical harmonic feature in F should have mean one. It is discussed in greater detail in Appendix B.

Given the radial model, the spherical harmonics, and the normalization coefficients, we can write the definition of the forward and backward passes of the Kernel. In the definition of Kernel, the radial model produces enough scalars to multiply each of the spherical harmonics from the filter but it also produces scalars which multiply the input and output features and their multiplicities. This introduces new indices l_{in}, l_{out} for spherical harmonic orders and index u for multiplicity. We write the forward pass of the kernel,

$$K_{ui\,vj}^{l_{out} l_{in}} = \sum_{l_f\;k} C_{ijk}^{l_{out} l_{in} l_f}\; Y_k^{l_f}\; R_{uv}^{l_{out} l_{in} l_f}\; n^{l_{out} l_{in}}, \tag{2.42}$$

and the backward pass,

$$\frac{\partial}{\partial R_{uv}^{l_{out} l_{in} l_f}} = \sum_{i\;j\;k} \frac{\partial}{\partial K_{ui\,vj}^{l_{out} l_{in}}} C_{ijk}^{l_{out} l_{in} l_f}\; Y_k^{l_f}\; n^{l_{out} l_{in}} \tag{2.43}$$

$$\frac{\partial}{\partial Y_k^{l_f}} = \sum_{l_{out}\;l_{in}\;ui\;vj} \frac{\partial}{\partial K_{ui\,vj}^{l_{out} l_{in}}} C_{ijk}^{l_{out} l_{in} l_f}\; R_{uv}^{l_{out} l_{in} l_f}\; n^{l_{out} l_{in}}. \tag{2.44}$$

Convolution

Once a Kernel has been calculated, the complete evaluation of G is finished in the Convolution by combining the existing features per convolved atom. Since the ideas were introduced in the Kernel, there isn't much else to say except see the equations for the forward,

$$G_{ui\,A}^{l_{out}} = \sum_{B\;l_{in}\;vj} K_{ui\,vj\,AB}^{l_{out} l_{in}}\; F_{vj\,B}^{l_{in}}, \tag{2.45}$$

and backward passes,

$$\frac{\partial}{\partial F_{vj\,B}^{l_{in}}} = \sum_{A\;l_{out}\;ui} \frac{\partial}{\partial G_{ui\,A}^{l_{out}}} K_{ui\,vj\,AB}^{l_{out} l_{in}} \tag{2.46}$$

$$\frac{\partial}{\partial K_{ui\,vj\,AB}^{l_{out} l_{in}}} = \frac{\partial}{\partial G_{ui\,A}^{l_{out}}} F_{vj\,B}^{l_{in}}. \tag{2.47}$$

$$\tag{2.48}$$

KernelConv

While splitting the Kernel and the Convolution into different operations allows for more flexibility, it also increases the memory footprint and decreases the speed of evaluation.

The Kernel has six indices without explicitly writing the atoms indices and the batch indices. In practice with the QM9 data set, the forward pass of the Kernel was over half a gigabyte of memory per layer in mini-batches of 12 with 30 atoms per batch. Even on gpus with ∼10.5 gigabytes of memory, this severely limited data throughput. This problem can be solved by avoiding saving the Kernel and instead calculating G in one step. That step is KernelConv. To get the 2x speed up, both the forward and backward passes needed to be custom implemented in PyTorch. The memory footprint was reduced such that it became possible to train on mini-batches of 30 molecules with 30 atoms each. Access this code in e3nn [17].

When the Kernel and the Convolution are combined into one operation, it is no longer possible to leave out the indices A, B; therefore, they return to the radial model R, spherical harmonics Y, normalization n, input features F, and output features G in this section. In the previous definitions they were always there, but since the summation did not depend on them, it is strictly more general to leave them out. One final index, Z, is still left out because batches never enter the summation. The forward pass is written,

$$G^{l_{out}}_{ui\,A} = \sum_{B\, l_f\, l_{in}\, vj\, k} C^{l_{out}l_{in}l_f}_{ijk}\; Y^{l_f}_{k\,AB}\; R^{l_{out}l_{in}l_f}_{uv\,AB}\; n^{l_{out}l_{in}}_{AB}\; F^{l_{in}}_{vj\,B}, \tag{2.49}$$

and the backward pass looks like,

$$\frac{\partial}{\partial F^{l_{in}}_{vjB}} = \sum_{A\, l_{out}\, l_f\, ui\, k} \frac{\partial}{\partial G^{l_{out}}_{ui\,A}} C^{l_{out}l_{in}l_f}_{ijk} Y^{l_f}_{kAB} R^{l_{out}l_{in}l_f}_{uv\,AB} n^{l_{out}l_{in}}_{AB} \tag{2.50}$$

$$\frac{\partial}{\partial R^{l_{out}l_{in}l_f}_{uv\,AB}} = \sum_{i\,j\,k} \frac{\partial}{\partial G^{l_{out}}_{ui\,A}} C^{l_{out}l_{in}l_f}_{ijk}\; Y^{l_f}_{k\,AB}\; n^{l_{out}l_{in}}_{AB} F^{l_{in}}_{vj\,B} \tag{2.51}$$

$$\frac{\partial}{\partial Y^{l_f}_{k\,AB}} = \sum_{l_{out}\, l_{in}\, ui\, vj} \frac{\partial}{\partial G^{l_{out}}_{ui\,A}} C^{l_{out}l_{in}l_f}_{ijk}\; R^{l_{out}l_{in}l_f}_{uv\,AB}\; n^{l_{out}l_{in}}_{AB} F^{l_{in}}_{vj\,B}. \tag{2.52}$$

Gated Block

The gated block introduces nonlinearities to e3nn. For the scalar features, i.e. $l_{out} = 0$, with nonlinearity σ, it works just like a multi-layer perceptron,

$$G^{0}_{ui\,A} \mapsto \sigma(G^{0}_{ui\,A}). \tag{2.53}$$

However, for the other features, a general nonlinearity is not an option since the operation will not be equivariant. We take advantage of the fact that multiplication by a scalar is equivariant and then use a nonlinearity on that scalar. Gated Block, therefore, needs access to the Kernel in order to calculate an extra scalar for every non-scalar output feature. The pair is multiplied together before being passed to the next layer. Explicitly, given another nonlinearity $\tilde{\sigma}$, for every non-scalar feature $G^{l_{out}\neq 0}_{ui}$, calculate an extra paired scalar $G^{0\,l_{out}}_{ui}$ in the Kernel then write,

$$G^{l_{out}\neq 0}_{ui} \mapsto \tilde{\sigma}(G^{0\,l_{out}}_{ui}) G^{l_{out}\neq 0}_{ui}. \tag{2.54}$$

Equivariance to SE(3)

To show the equivariance of e3nn to SE(3), we consider `KernelConv` and `GatedBlock` separately. Given a rotation $r \in \mathrm{SO}(3)$, translation $t \in \mathbb{R}^3$, atom positions x, and input features $F^{l_{in}}_{vj\,B}$, `KernelConv` transforms like

$$G^{l_{out}}_{ui\,A}(x, F^{l_{in}}_{vj\,B}) \mapsto G^{l_{out}}_{ui\,A}(\rho(r)^{-1}(x-t), D^{l_{in}}_{jj'}(r)\,F^{l_{in}}_{vj'\,B}) \tag{2.55}$$

Recall that $R^{l_{out}l_{in}l_f}_{uv\,AB}$ only depends on the pairwise distances, $\|x_A - x_B\|$, which implies that R is invariant to translation and rotation because $\|\rho(r)^{-1}z\| = \|z\|$ and $(x_A-t)-(x_B-t) = x_A - x_B$. $C^{l_{out}l_{in}l_f}_{ijk}$ and $n^{l_{out}l_{in}}_{AB}$ both do not have a dependence on the input so they are invariant to translation and rotation. The spherical harmonics are calculated on a unit vector of pairwise displacements, therefore the evaluation is invariant to translation, i.e. $Y^{l_f}_{k\,AB}(z-t) = Y^{l_f}_{k\,AB}(z)$. By the properties of spherical harmonics from (2.28), $Y^{l_f}_{k\,AB}(\rho(r)^{-1}z) = \sum_{k'} D^{l_f}_{kk'}(r) Y^{l_f}_{k'\,AB}(z)$. Using the defining property of the Clebsch-Gordan decomposition followed by relabeling the primed indices, we can rewrite (2.55) after transformation and rotation as,

$$\begin{aligned}
&\sum_{B\,l_f\,l_{in}\,vjj'\,kk'} C^{l_{out}l_{in}l_f}_{ijk}\; D^{l_f}_{kk'}(r)\; D^{l_{in}}_{jj'}(r)\; Y^{l_f}_{k'\,AB}\; R^{l_{out}l_{in}l_f}_{uv\,AB}\; n^{l_{out}l_{in}}_{AB}\; F^{l_{in}}_{vj'\,B} \\
&= D^{l_{out}}_{i'i}(r) \sum_{B\,l_f\,l_{in}\,vj'\,k'} C^{l_{out}l_{in}l_f}_{ij'k'}\; Y^{l_f}_{k'\,AB}\; R^{l_{out}l_{in}l_f}_{uv\,AB}\; n^{l_{out}l_{in}}_{AB}\; F^{l_{in}}_{vj'\,B} \\
&= D^{l_{out}}_{ii'}(r)\; G^{l_{out}}_{ui'\,A}(x, F^{l_{in}}_{vj\,B}),
\end{aligned} \tag{2.56}$$

which proves the equivariance of `KernelConv` to SE(3). Now `GatedBlock` is rather simple. The case that $Y^{l_{out}=0} = 1$, implies that the angular part is invariant to input entirely. We already know that the radial part is invariant to rotation and translation. For the other orders, we know that multiplication by a scalar is equivariant to rotation and,

$$\begin{aligned}
\tilde{\sigma}(G^{0\,l_{out}}_{ui})G^{l_{out}\neq 0}_{ui} \mapsto\; & \tilde{\sigma}(G^{0\,l_{out}}_{ui})G^{l_{out}}_{ui\,A}(\rho(r)^{-1}(x-t), D^{l_{in}}_{jj'}(r)\;F^{l_{in}}_{vj'\,B}) \\
&= \tilde{\sigma}(G^{0\,l_{out}}_{ui})\; D^{l_{out}}_{ii'}(r)\; G^{l_{out}}_{ui'\,A}(x, F^{l_{in}}_{vj'\,B}) \\
&= D^{l_{out}}_{ii'}(r)\; \tilde{\sigma}(G^{0\,l_{out}}_{ui})\; G^{l_{out}}_{ui'\,A}(x, F^{l_{in}}_{vj'\,B})
\end{aligned} \tag{2.57}$$

from the last proof and commutative properties of scalars. Finally, since a single layer of convolution and gated block is equivariant, an entire network is equivariant by induction.

2.4.2 QM9 Output Layer

Neural networks operate best when they are predicting values close to one. For this reason, we perform a decomposition of the target value such that the regression network's output fits this criteria. The implementation of this part of the network was handled by schnetpack [41]. The decomposition utilizes the reference values in the QM9 data set so the network starts with a good guess and predicts a perturbation from that guess. The network's prediction is decomposed into a reference bias, an atom-wise sum from the Table 2.3, and a scaled contribution from each atom.

For any target in Table 2.3 and atom in qm9, we can create a map from element Z and prediction column C to the corresponding reference value R called $ref(Z, C)$. For example, $ref(\mathrm{H}, \mathrm{U}_0) = -0.5$ Hartree. Given a training set of M molecules indexed by

Element	ZPVE Hartree	U (0 K) Hartree	U (298.15 K) Hartree	H (298.15 K) Hartree	G (298.15 K) Hartree	Heat Capacity Cal/(Mol Kelvin)
H	0.000	-0.500	-0.499	-0.498	-0.511	2.981
C	0.000	-37.847	-37.845	-37.844	-37.861	2.981
N	0.000	-54.584	-54.582	-54.582	-54.599	2.981
O	0.000	-75.065	-75.063	-75.062	-75.080	2.981
F	0.000	-99.719	-99.717	-99.716	-99.734	2.981

Table 2.3: Table is adapted from the qm9 paper [11].

$m \in \{1, 2, ..., M\}$ each with A_m atoms indexed by $a_m \in \{1, 2, ..., A_m\}$ with a corresponding element Z_{a_m}, we write the reference bias

$$p_m = \sum_{a_m=1}^{A_m} ref(Z_{a_m}, C). \tag{2.58}$$

To further our decomposition consider the target regression value for a certain molecule t_m. From the ground truth, we can write the atom-wise deviation from the reference value,

$$\tilde{t}_m = \frac{t_m - p_m}{A_m}. \tag{2.59}$$

By gathering statistics from the training data on this $\tilde{t}_m$, we will achieve our goal of normalizing the output of the regression network to something near one. Let $\bar{\tilde{t}}$ and $\sigma_{\tilde{t}}$ be the mean and standard deviation of $\tilde{t}_m$ over molecules respectively. Given the atom-wise output of a regression network $\mathcal{R}_{a_m}$, we predict the ground truth target $\hat{t}_m$ by

$$\begin{aligned}\hat{t}_m &= p_m + \sum_{a_m=1}^{A_m} \left(\bar{\tilde{t}} + \sigma_{\tilde{t}} \mathcal{R}_{a_m}\right) \\ &= p_m + A_m \bar{\tilde{t}} + \sigma_{\tilde{t}} \sum_{a_m}^{A_m} \mathcal{R}_{a_m} \\ &= p_m + \frac{A_m}{M} \sum_{n=1}^{M} \tilde{t}_n + \sigma_{\tilde{t}} \sum_{a_m}^{A_m} \mathcal{R}_{a_m}.\end{aligned} \tag{2.60}$$

Tetris Experiment

The tetris experiment empirically demonstrates that e3nn can learn on data in any orientation and trivially generalize to any other orientation without data augmentation [40]. It is presented here to emphasize something different than its initial formalization, specifically, that some predictions require the presence of rotating internal features to be accurate, even when the output prediction itself is invariant to rotation.

It is a simple supervised learning problem, given the positions of the 3d tetris blocks: Identify the block from the eight possible classes. The network predicts a target array $\hat{t} \in \mathbb{R}^8$, such that $\hat{t}_i > 0$ and $\sum_i \hat{t}_i = 1$ where every $\hat{t}_i$ is a scalar quantity which is invariant to rotation. The correct class is identified using a one-hot scalar array and the network learns by minimizing $\|\hat{t} - t\|^2$.

There are two chiral pieces in the problem as seen in Figure 2.11. The radial part of the filter is determined using pairwise distances, which are invariant under the parity transformation. (Parity is a transformation which takes a chiral piece to its mirror image). If the contribution from the spherical harmonics to the learned filter is only of order zero, i.e. $Y^0_m = 1$, then the model only has access to pairwise distance information. The consequence of this is that, to a network which only learns scalar features, the chiral pieces are indistinguishable; however, for a network which has spherical harmonic features of order one or greater, they can be uniquely identified. This is an example of when the target is a scalar feature invariant to rotation, namely, probability; however, a network which only uses scalars as internal representation cannot solve the learning problem exactly.

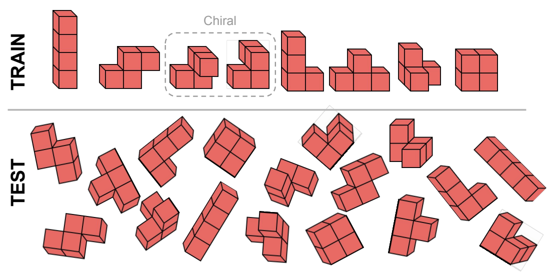

Figure 2.11: Images of the blocks in the 3d tetris problem. The training set is in a single fixed orientation but the network is evaluated on a test set which is presented at a random rotation. Networks invariant to rotation will not be able to distinguish the chiral pieces.

The evidence for the claim can be seen in Figure 2.12. The plots indicate that networks which contain internal spherical harmonic features order >0 are able to correctly predict the identity of the shapes with 100% accuracy; however, training does not improve the accuracy of a scalar-only network past 7/8. This core result will resurface in Section 3.3 when the network learns to predict the dipole moment.

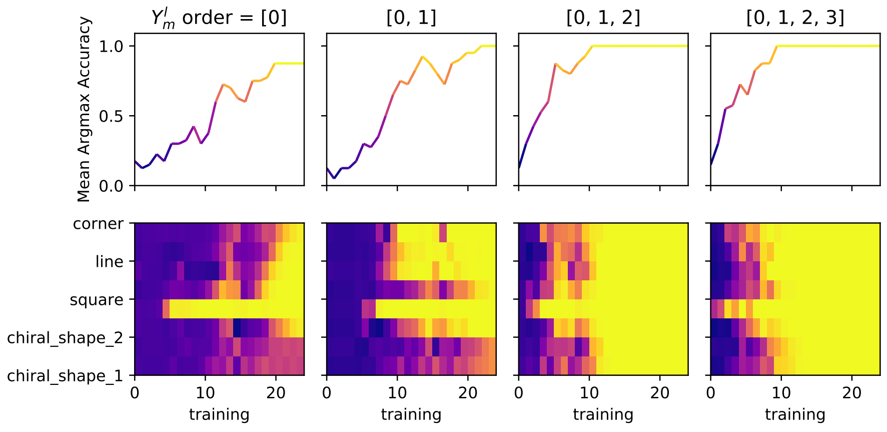

Figure 2.12: Results of the tetris task are shown. Each column identifies the rotation order of the network's spherical harmonic features. The top row plots the average accuracy of the network on the test set versus training. The network makes a prediction using the argmax function over the probability array. Networks with spherical harmonics of order one or greater complete the task accurately, but the network with scalar internal features cannot distinguish the chiral shapes –no matter the length of the training. The second row of color plots clarifies the performance per class. Within each color plot, the y-axis is divided into rows labeled with shapes. Each row of color represents how accurately the network can identify that shape as it trains. Evidently, some shapes are easier to identify than others.

Chapter 3

Experiments

The benchmark results of e3nn on the QM9 data set are the core products of this work.

3.1 Comparison with SchNet-like Architecture

The primary difference between e3nn and SchNet is the inclusion of features which are not invariant to rotation. SchNet is strictly a special case of e3nn in which there are no spherical harmonic features greater than order zero. In order to determine the value of higher-order spherical harmonic features it is natural to compare e3nn to SchNet across different hyperparameter ranges and levels of training. Using the hyperparameters reported in schnetpack [41] as a starting point, this section compares the effects of hyperparameter variation on networks with higher-order spherical harmonic features to those without.

To easily refer to the rotation properties of the internal features, unless stated otherwise, we use the shorthand L·Net where the · is filled with the highest order spherical harmonic included, i.e. L2Net implies that spherical harmonics of order two, one, and zero are all included in the network. Given this shorthand, a SchNet-like model would be called an L0Net. Another way of referring to the rotation properties is to say that an L0Net has internal features which do not rotate while L1Nets and above have rotating internal features. The basic architecture of an example L1Net is diagrammed in Figure 3.1. For further "default" hyperparameter values of L0Net and L1Net, consult Table 3.1. It is mentioned specifically whenever values differ from this table.

As a starting experiment, an L0Net and an L1Net were trained to reduce the batch-wise mean squared error between the prediction and the target for 55 epochs using the Adam optimizer [29] with default parameters. The data was divided, like in the SchNet paper [16], into a training set of 10900 molecules, validation set of 1000 molecules, and a test set containing all remaining molecules. The same data split and randomization seed was used throughout all experiments. In this section, each target was trained using a different set of initial weights and only trained to predict a single property at a time.

The results of the experiment are presented in Figure 3.2 where the mean average error of L0Net and L1Net on the validation set is plotted against the number of epochs. The performance of the two models against the test data set is shown in Table 3.2. Throughout training, the most obvious finding is that the prediction on the dipole moment is significantly improved by the L1Net compared with the L0Net. Other targets including enthalpy (H), energy (U & U0), isotropic polarizability (alpha), LUMO, and the electronic spatial extent (r2) are slightly more accurate; however, the difference looks like a perturbation rather than a seismic difference in predictive power.

Recall bias-variance decomposition from Section 2.1.1. When limited to only one training example, we are effectively estimating the expected squared error using a single sample, which does not tell us anything about the variance in the model given these hyperparameters. Although this is not ideal, it is common practice in the machine learning literature and given technological constraints, i.e. access to limited gpus and time, the variance estimation was not possible. (The calculations in Figure 3.2 would take 24 days on a single Nvidia GeForce GTX 1080 Ti, even with the 2x speedup from the author's formulation of `KernelConv` in Section 2.4.1.) Given the unknown noise characteristics, it is difficult to say definitively whether the L1Net is outperforming the L0Net on the values other than the dipole moment; however, the consistency of higher performance across different targets implies a prediction improvement from L1Net which may not be attributable to noise. It is worth mentioning that **this dipole prediction accuracy is currently state of the art**; however, the accuracy will be further improved in the following section after a hyperparameter search.

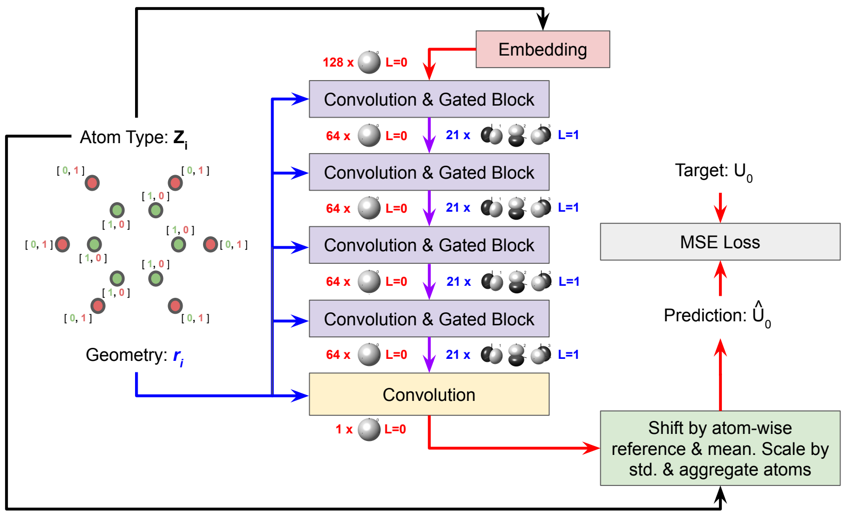

Figure 3.1: Diagram of a sample qm9 regression network "L1Net" designed to train on a single target. Atom type information, encoded as a one-hot scalar array, passes through a learned embedding and the into the network as a scalar feature. The geometry is fed to the filter in every convolutional layer. Each feature is then transformed by the gated block. The color of the arrows indicates what type of data is transmitted. Black is a one-hot array, red implies scalar information, blue means a feature which rotates, purple is a direct sum of both scalar and rotating data. The last layer of the network does not apply a nonlinearity in order to mimic the output layer of other neural networks. Finally, the output of the network is scaled by statistics and shifted by reference atomization values and statistical deviations from atomization values, i.e. (2.60), yielding the final prediction. That prediction is compared with the ground truth with the mean squared error loss. The hyper-parameters written in the diagram, like the target and number of features, are merely an example. The important hyper-parameters will be specified when discussing the results of training, for example in Table 3.1.

Hyperparameter	L0Net	L1Net
Batch Size	16	16
Learning Rate	10^{-4}	10^{-4}
Size of Embedding	128	128
Representation	$128 \times Y_m^0$	$64 \times Y_m^0 \oplus 21 \times Y_m^1$
Convolutional Layers	5	5
Gated Block Layers	4	4
Residual Network	True	True
Radial Basis	Cosine ϕ_C	Cosine ϕ_C
Number of Radial Bases	50	50
Radial Maximum	10.0 Å	10.0 Å
Radial MLP Layers	2	2
Radial MLP Neurons	128	128

Table 3.1: These hyperparameters are used throughout the section unless stated otherwise. A few notes: The gated blocks follow the first four convolutions. The Radial Maximum implies the maximum center for a radial basis, therefore it is possible to convolve over atoms farther away than the Radial Maximum. The basis determines how far. The design principle in choosing the size of the representation was to maintain approximately the same number of components to SchNet, which has 128 element scalar arrays as features.

Target	L0Net	L1Net
μ (D)	0.053	**0.016**
α (a_0^3)	0.027	**0.025**
ϵ_{HOMO} (eV)	0.085	**0.080**
ϵ_{LUMO} (eV)	0.074	**0.064**
ϵ_{gap} (eV)	0.113	**0.106**
$\langle R^2 \rangle$ (a_0^2)	0.647	**0.482**
zpve (eV)	0.003	0.003
U_0 (eV)	0.052	**0.050**
U (eV)	**0.049**	0.050
H (eV)	0.074	**0.047**
G (eV)	0.050	**0.048**
C_v ($\frac{\text{cal}}{\text{molK}}$)	**0.070**	0.073

Table 3.2: This table presents the mean average error of L0Net and L1Net on the test data. Without significant hyperparameter optimization, the dipole moment result is already state of the art. a_0 is defined to be the bohr radius. The results are reported after fifty-five epochs of training.

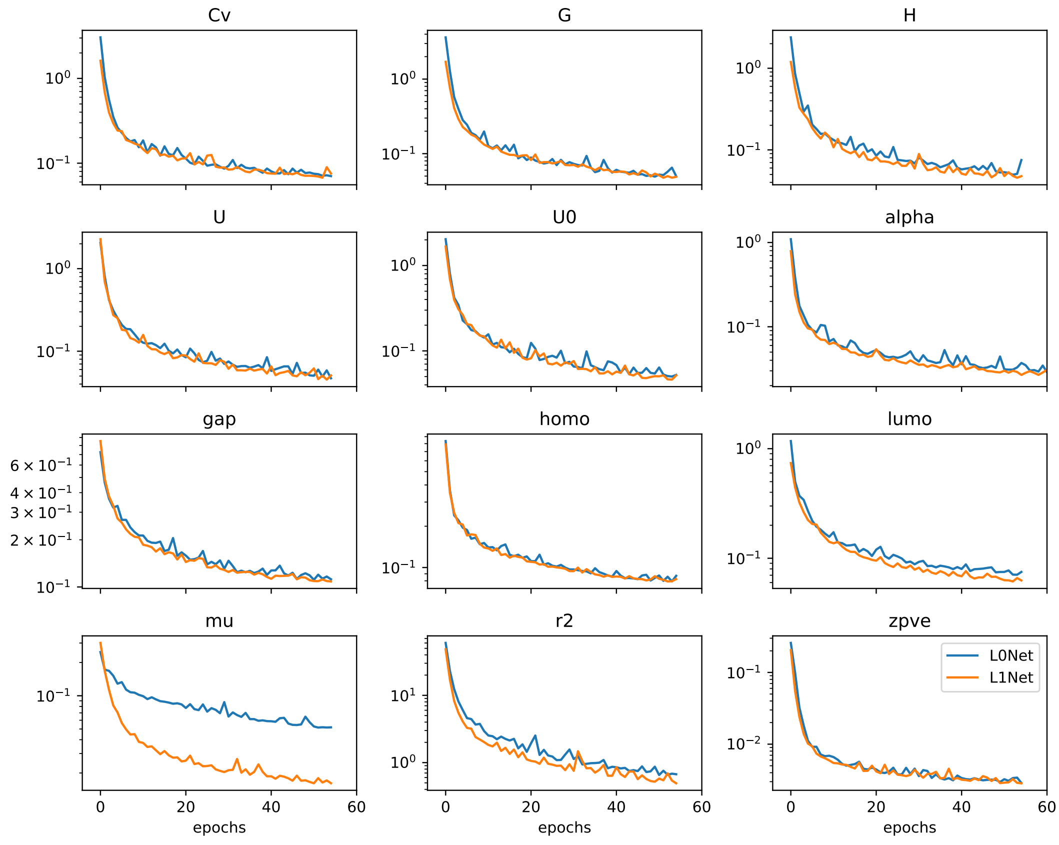

Figure 3.2: The mean average error on the validation set is plotted against the number of epochs with a logarithic vertical axis for various targets. The blue line has no rotating internal features while the orange line contains spherical harmonic features of up to order one. The performance of L1Net is qualitatively superior on the dipole moment and possibly superior on enthalpy (H), energy (U & U0), isotropic polarizability (alpha), LUMO, and the electronic spatial extent (r2); however, without error bars it is hard to tell. The hyperparameters for these models can be found in Table 3.1.

3.1.1 Effects of Higher Rotation Orders on Dipole Moment and Energy

Including the first order spherical harmonic in the internal features of L1Net significantly improved the dipole moment prediction but only somewhat improved the energy prediction. Would the inclusion of higher order spherical harmonics further increase the prediction accuracy? The answer to that question can be seen in the plot of the mean average error on the validation set in Figure 3.3.

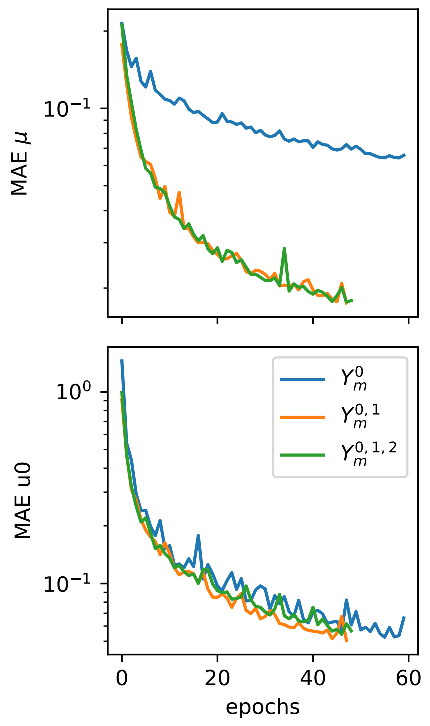

Figure 3.3: The mean average error on the validation set for dipole moment and energy at 0K versus the number of training epochs. The blue line has no rotating internal features, the orange line includes spherical harmonics up to order one, the green line includes spherical harmonics up to order two. The first order spherical harmonic seems very important to accurately predict the dipole moment but the inclusion of the second order spherical harmonic doesn't have a qualitatively stronger effect. In the energy, the story is mostly the same for all different models. There is a level of symmetry after which higher rotation orders give diminishing returns.

The blue, orange, and green lines correspond to L0Net, L1Net, and L2Net respectively. L2Net's hyperparameters are not listed in the table above, but they would be the same except for representation which is $42 \times Y_m^0 \oplus 14 \times Y_m^1 \oplus 8 \times Y_m^2$. Since each spherical harmonic has $2l+1$ components, the sum of all of the multiplicities of spherical harmonic components is approximately the same size as SchNet's 128 feature scalar arrays.

Depending on the target, rotating internal features can have a positive effect on accuracy; however, qualitatively, the performance of L2Net is not better, on either target, than L1Net. The benefits of including higher order spherical harmonics have diminishing returns when predicting the dipole moment. From this experiment, it appears important to include the first order spherical harmonic but not necessarily the second one.

3.1.2 Comparison of Radial Bases

The different radial bases were introduced in Section 2.4.1. In DimeNet, the radial Bessel basis contributed to strong results on QM9 benchmarking [20]. How does e3nn respond to the choice of radial basis? Recall, the cosine basis is sharply peaked and evaluates to zero for all but a small range of the positive real numbers; the Gaussian basis has a broader peak and no part of the positive reals evaluates to zero; the Bessel basis has multiple peaks and a cutoff, outside of which there are no contributions. L0Net and L1Net were trained to predict the dipole moment using all three basis functions and the results are plotted in Figure 3.4.

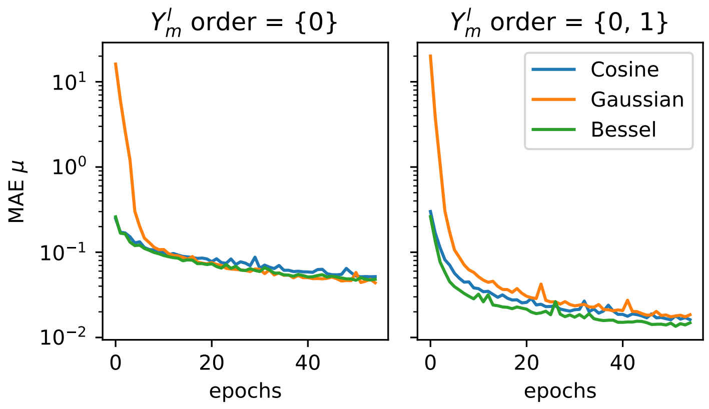

Figure 3.4: The mean average error in the prediction of the dipole moment is plotted against the number of training epochs. The blue, orange, and green lines correspond with the cosine, Gaussian, and Bessel bases respectively. The left plot corresponds to the L0Net while the plot on the right corresponds to an L1Net. With these hyperparameters, the choice of basis function does not have a strong effect on the L0Net; however, the effect is more pronounced on the L1Net with the Bessel basis performing best.

Given a relatively high number of radial bases along with a long max radius (long because most atoms can see all other atoms at 10.0 Å), the effect of basis choice is not very pronounced on L0Net. The Bessel basis performed noticeably better in the L1Net. The choice of radial hyperparameters inducing qualitatively different training dynamics between networks with different rotation properties implies that there might be a connection worth investigating in the future. Notably, despite the strong performance of the Bessel basis on this task, it was not selected as the radial basis in any of the top performing networks during a random hyperparameter search in Section 3.2.1.

3.1.3 Effects of Batch Size

Even with powerful multiprocessing gpus, it takes a very long time to calculate a gradient for every example in a large data set. For this reason, it is common to split the training into mini-batches where, after every mini-batch, the parameters are updated. (Mini-batches are often referred to as batches in this document.) It is a middle ground between seeing all of the training data before making a parameter update and updating with every training example. In addition to the increase of parallelism, there are also theoretical motivations for using large batch sizes. Consider, if an estimator were shown a number of similar training examples in a row, it may perturb training away from a generalized solution. Larger batch sizes contain more of the variance of the data set, which ought to be reflected in the parameter updates.

Despite the theoretical motivation to use a large batch size, there is empirical evidence that using smaller batch sizes can decrease the generalization error [42, 43, 44]. The gist of the argument is: By increasing the batch size, one effectively slows the traversal of parameter space. Large batches have large variance in the magnitude of the gradient updates, implying that large batches can yield very small steps in parameter space. Meanwhile, reducing the number of parameter update steps per epoch by increasing the batch size leads to slower traversal as well.

The effects of batch size on the training of L0Net and L1Net is shown in Figure 3.5. In L1Net, it was noticeable that using larger batch sizes increased the generalization error. Whether this can be attributed to the gradient arguments from the papers above is left for future work. However, this insight was utilized in determining the batch size hyperparameter range in the random hyperparameter search.

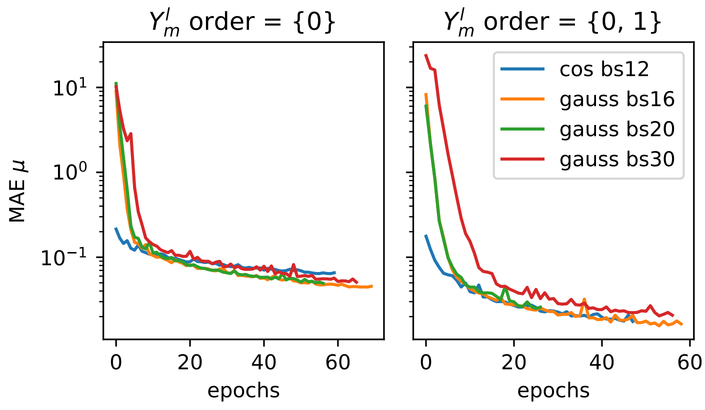

Figure 3.5: The mean average error in the prediction of the dipole moment is plotted against the number of training epochs. The different colored lines correspond to different batch sizes. The Gaussian basis was used for all colors, except blue which used the cosine basis. The left plot corresponds to the L0Net while the plot on the right corresponds to an L1Net. Although the effect is smaller on L0Net, larger batch sizes increased the generalization error for a given epoch. The greatest difference is between the 30 molecules per batch option compared with the others where the 30 molecule batches performed worst. (If one removes the cosine basis, this is true across both plots for all epochs.) Reducing the batch size may reduce the generalization error, but it also comes at the cost of slower calculation.

3.2 e3nn with a Multi-Layer Perceptron Output

In order to offer more flexibility on multi-target training, an alternative architecture was developed which divides the network into a featurization section followed by an output section. Each output block receives a copy of the same learned featurization; however, the output blocks do not share gradients or other information. This allows for each output block to transform the learned input features in parallel, each predicting a single target; thereby, the whole network makes predictions on multiple targets. The network design is depicted in Figure 3.6.

The featurization section is much like the network depicted above, i.e. an embedding followed by layers of Convolutions and Gated Blocks. A combination of order zero and order one spherical harmonic features are copied and passed to each output block. Each output block then passes those features through Convolutions and Gated Blocks and calculates an array of scalar features. Since they are scalar features, we can pass them through a multi-layer perceptron with a rectified linear unit activation without breaking equivariance (invariance in this case). The last layer of that multi-layer perceptron predicts a single scalar using a linear weighted sum of the previous layer's features, which is then passed to the shift and scale operation as seen in Section 2.4.2.

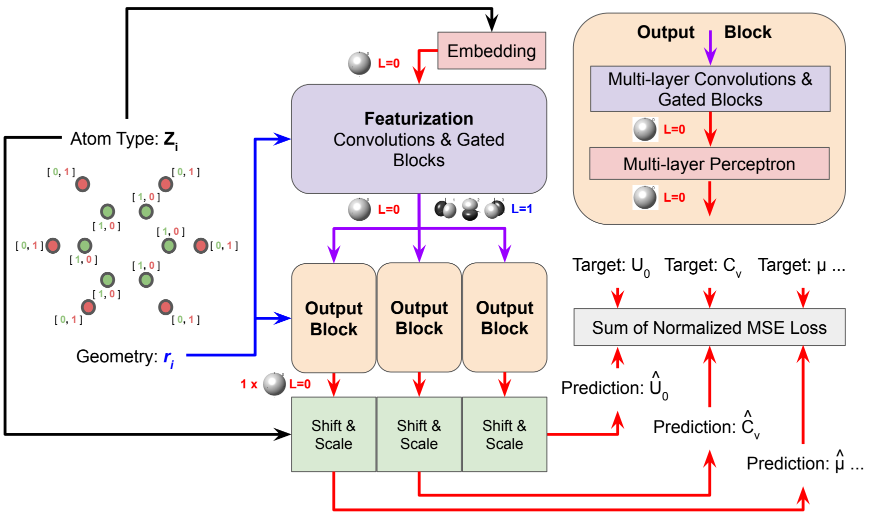

Figure 3.6: Diagram of the network designed for multi-target training. This design divides the featurization section from the output section. All output blocks are fed by the featurization layers, but information does not flow between output blocks. The color scheme and general layout follows the other network depicted in Figure 3.1. One of the only differences between the layers of this network and the previous one, is that the output layer has a multi-layer perceptron as the last layer. This is possible without breaking rotational equivariance because only scalar information is passed to the multi-layer perceptron.

The loss calculation is different from other learning problems because we attempt to normalize the losses across targets. Since all targets are equally important, we normalize their variance to one based off of statistics from our training data. This formulation depends on the assumption that each output block predicts mean zero at initialization. Recall, (2.60), the prediction is written,

$$\hat{t}_m = p_m + A_m \bar{\tilde{t}} + \sigma_{\tilde{t}} \sum_{a_m}^{A_m} \mathcal{R}_{a_m}. \tag{3.1}$$

Therefore, using the notation from Section 2.4.2, we write the loss using the the total molecule-wise offset $s_m = p_m + A_m \bar{\tilde{t}}$. The molecule-wise loss for target t_m looks like,

$$\mathcal{L}_m(t_m, \hat{t}_m) = \left(\frac{t_m - s_m}{\sigma_{\tilde{t}}} - \frac{\hat{t}_m - s_m}{\sigma_{\tilde{t}}}\right)^2 = \frac{1}{\sigma_{\tilde{t}}^2}(t_m - \hat{t}_m - 2s_m)^2. \tag{3.2}$$

For a batch of molecules M and pairs of targets with corresponding predictions $\{(t_m, \hat{t}_m) : m \in M\}$, the total loss is calculated by

$$\frac{1}{M} \sum_{(t_m, \hat{t}_m)} \sum_{m} \mathcal{L}_m(t_m, \hat{t}_m). \tag{3.3}$$

Although it is possible to train a model against all targets at the same time using this methodology, it is often much more difficult to achieve simultaneously good performance across targets.

3.2.1 Multi-target Hyperparameter Search

Even considering the potential difficulty of multi-target training, the computational benefits of training on all targets at the same time is very appealing for a hyperparameter search. The search would be organized by first finding an architecture which performs "well" on the multi-target task. The best performing architecture would then be trained on each target individually, thus eliminating the difficulty of multi-target training. This scheme rests on the assumption that an architecture which performs well on all targets simultaneously will also perform well on each target individually. This approach was utilized in finding the best performing models.

Finding the best hyperparameters is difficult because determining their quality requires training the network to convergence, which is expensive. The two most straightforward methods for finding hyperparameters are grid search and random search. To perform a grid search, one selects ranges of parameters and tests every possible combination of parameters within this multidimensional grid up to a certain granularity. Given n hyperparameters and m grid values per hyperparameter, this scales like $O(m^n)$. In our problem, this is a completely unacceptable scaling. Therefore, random search is utilized, where one draws hyperparameter values at random from within the ranges specified. Although it is less comprehensive, it still explores the possible configurations without taking excessively large amounts of time and computational power.

Along with numerical hyperparameters, the choice of radial basis is determined by selecting one of the cosine, Gaussian, and Bessel bases with one third probability for each. Otherwise, the ranges for the random hyperparameter search are shown in Table 3.3. The representation, i.e. amount of order zero spherical harmonics and order one spherical harmonics per convolutional layer, is determined like so: Let $0 \leq p \leq 1$ be a random number and C is the number of components, the multiplicity of order zero R_0 and order one R_1 spherical harmonics are determined by

$$R_0 = pC, \quad R_1 = (1 - p)C. \tag{3.4}$$

The hyperparameter search involved sampling forty different sets of hyperparameters from the ranges in Table 3.3 and doing multi-target training for ten epochs with each set of hyperparameters. The test set performance for each of the forty models was compared. Two models performed well on multiple targets: ARandNet outperformed all other hyperparameter sets on five targets while BRandNet outperformed other hyperparameter sets on four disjoint targets. The specific hyperparameters for those models can be found in Table 3.4. ARandNet and BRandNet were selected for further single-target training.

Hyperparameter	Minimum	Maximum
Batch Size	8	25
Learning Rate	10^{-6}	3×10^{-1}
Size of Embedding	80	144
Featurization Components (FC)	80	144
Featurization Representation	(FC) randomly divided	between Y_m^0 and Y_m^1
Featurization Conv & GBs	2	5
Residual Network	True	True
Radial Basis	ϕ_C, ϕ_G, ϕ_B	ϕ_C, ϕ_G, ϕ_B
Number of Radial Bases	25	100
Radial Maximum	1.2 Å	30.0 Å
Radial MLP Layers	1	3
Radial MLP Neurons	80	144
Output Components (OC)	64	128
Output Representation	(OC) randomly divided	between Y_m^0 and Y_m^1
Output Conv & GBs	1	2
Output MLP Layers	1	3
Output MLP Neurons	80	144

Table 3.3: The ranges of hyperparameters for the random hyperparameter search are written in this table. Featurization Representation and Output Representation is determined by (3.4).

Hyperparameter	ARandNet	BRandNet
Batch Size	16	18
Learning Rate	0.0065	0.0194
Size of Embedding	93	80
Featurization Components (FC)	107	94
Featurization Representation	$20 \times Y_m^0 \oplus 29 \times Y_m^1$	$94 \times Y_m^0 \oplus 1 \times Y_m^1$
Featurization Conv & GBs	2	4
Residual Network	True	True
Radial Basis	ϕ_C	ϕ_G
Number of Radial Bases	84	51
Radial Maximum	11.1 Å	2.87 Å
Radial MLP Layers	2	1
Radial MLP Neurons	101	100
Output Components (OC)	89	84
Output Representation	$5 \times Y_m^0 \oplus 28 \times Y_m^1$	$48 \times Y_m^0 \oplus 12 \times Y_m^1$
Output Conv & GBs	1	1
Output MLP Layers	1	2
Output MLP Neurons	67	51

Table 3.4: The hyperparameters of two networks determined randomly. On the multi-target task, ARandNet had the lowest mean average error across five targets, while BRandNet had the lowest mean average error across four (different) targets. Since there are twelve total targets, these two architechtures were nearly tied for best performance on the multi-target task.

3.3 Regression on QM9 Properties

The SchNet-like and e3nn with MLP Output network architectures were described in sections 3.1 and 3.2. The multi-target hyperparameter search was explained in Section 3.2.1 along with the results for the best performing set of hyperparameters. Finally, we consider the performance of L0Net, L1Net, ARandNet, and BRandNet against other competitive models in the field.

The other neural networks have already been discussed, but as a refresher: SchNet [16, 41] is a neural network which performs continuous convolutions over a graph of euclidean distances. It uses scalar features and is completely invariant to translation, rotation, or parity transformations. Cormorant [18] is quite similar to e3nn in that it offers spherical harmonic features which can rotate; however, it does not include the gated block, i.e. no gated nonlinearities. Furthermore, it includes another interaction between atoms, they call a two-body interaction, which is absent from e3nn. DimeNet [20] is another graph model but includes angular information by means of calculating scalars which depend on the bond angles in the problem. In this way, they include angular information which rotates along with the molecule.

A comprehensive list of results on all targets is presented in Table 3.5. The random hyperparameter search yielded ARandNet which now holds the state of the art prediction on three of twelve QM9 targets: dipole moment, isotropic polarizability, and electronic spatial extent.

Target	L0Net	L1Net	ARandNet	BRandNet	schnetpack	Cormorant	DimeNet
μ (D)	0.053	0.016	**0.009**	0.136	0.021	0.038	0.0286
α (a_0^3)	0.027	0.025	**0.013**	0.038	0.124	0.085	0.0469
ϵ_{HOMO} (meV)	85	80	46.246	232.226	47	34	**27.8**
ϵ_{LUMO} (meV)	74	64	34.351	1012.791	39	38	**19.7**
ϵ_{gap} (meV)	113	106	68.381	903.469	74	61	**34.8**
$\langle R^2 \rangle$ (a_0^2)	0.647	0.482	**0.132**	15.434	0.158	0.961	0.331
zpve (meV)	3	3	1.644	2.695	1.616	2.027	**1.29**
U_0 (meV)	52	50	13.94	39.344	12	22	**8.02**
U (meV)	49	50	13.72	862.5	12	21	**7.89**
H (meV)	74	47	13.93	120.063	12	21	**8.11**
G (meV)	50	48	14.597	33.233	13	20	**8.11**
C_v ($\frac{\text{cal}}{\text{molK}}$)	0.070	0.073	0.031	0.172	0.034	0.026	**0.0249**

Table 3.5: L0Net and L1Net were defined in Section 3.1 and trained for 55 epochs. The architectures from the random hyperparameter search, ARandNet and BRandNet, are listed in the next two columns and were trained for 100 epochs. The other benchmarks are taken from schnetpack [41], Cormorant [18], and DimeNet [20] papers. Details about their implementations and hyperparameters can be found in the corresponding publications. The first random hyperparameter model, **ARandNet, holds state of the art predictions on this data set for the targets dipole moment $\boldsymbol{\mu}$, isotropic polarizability $\boldsymbol{\alpha}$, and electronic spatial extent $\boldsymbol{\langle R^2 \rangle}$.** a_0 is defined to be the bohr radius.

3.3.1 Dipole Moment

To determine why e3nn performs so well on the dipole moment, let us consider how to calculate the magnitude of the dipole moment of a charge distribution. For a continuous distribution of charge contained in a volume V with charge density ρ, the dipole moment is defined,

$$\mathbf{p}(\mathbf{r}) = \iiint_V \rho(\mathbf{r}_0)(\mathbf{r}_0 - \mathbf{r})d^3\mathbf{r}_0, \tag{3.5}$$

where $\mathbf{r}$ denotes the position of observation and $d^3\mathbf{r}_0$ an infinitesimal volume element. If we model the interaction between two dipoles as the interaction between two pairs of positively and negatively charged Dirac delta peaks separated by an small displacement, we can write the integral as a sum,

$$\mathbf{p}(\mathbf{r}) = \sum_{i=1}^{4} q(\mathbf{r}_i - \mathbf{r}) = q(\mathbf{r}_A - \mathbf{r}_B + \mathbf{r}_C - \mathbf{r}_D) = \mathbf{p}_1 + \mathbf{p}_2, \tag{3.6}$$

where particles A and C have charge q while particle B and D have charge -q, and the pairs A and B have dipole moment $\mathbf{p}_1$ while the pairs C and D have dipole moment $\mathbf{p}_2$. The result is that the dipole moment is independent of position of observation, i.e. $\mathbf{p}(\mathbf{r}) = \mathbf{p}$. See Figure 3.7.

Our goal was to calculate the magnitude of the net dipole moment. For explanatory purposes, we instead calculate the squared magnitude, which can be written,

$$p^2 = \|\mathbf{p}_1 + \mathbf{p}_2\|^2 = p_1^2 + 2p_1p_2\cos\theta_{12} + p_2^2, \tag{3.7}$$

where $\|\mathbf{p}\| = p$ and $\frac{\mathbf{p}_1 \cdot \mathbf{p}_2}{p_1 p_2} = \cos\theta_{12}$.

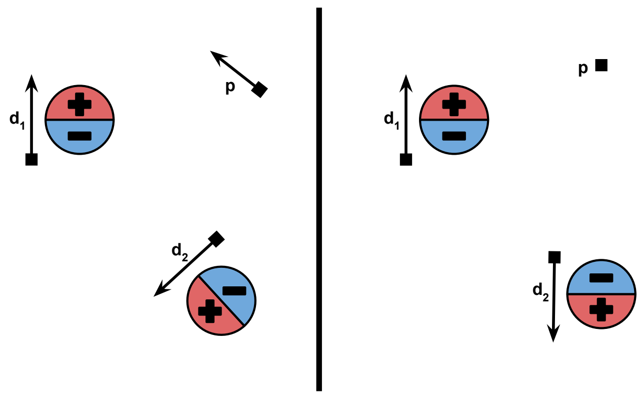

Figure 3.7: The total dipole from the addition of constituent dipoles depends on their relative orientation. Consider the addition of two constituent dipole moments with equal magnitude. In the left pane, the sum of the two constituents yields a non-zero net dipole, while in the right pane they cancel out. This is an interaction between two vector quantities; it cannot be captured across relative orientations as a function of the constituent's (scalar) magnitudes alone.

It is already apparent that modeling this equation without a dependence on θ_{12} will inevitably lead to irreducible error. Let f be the output of an estimator which does not depend on θ_{12} but estimates the squared magnitude of the dipole moment. We can write the expectation value of the mean squared error between this prediction and the ground truth,

$$\begin{aligned}
\mathbb{E}[(f-p^2)^2] =& \mathbb{E}[f^2 - 2fp^2 - p^2] \\
=& \mathbb{E}[f^2 - 2f(p_1^2+p_2^2) - 4fp_1p_2\cos\theta_{12} \\
& + (p_1^2+p_2^2)^2 + 4(p_1^2+p_2^2)p_1p_2\cos\theta_{12} + 4p_1^2p_2^2\cos^2\theta_{12}] \\
=& f^2 - 2f(p_1^2+p_2^2) - 4fp_1p_2\mathbb{E}[\cos\theta_{12}] \\
& + (p_1^2+p_2^2)^2 + 4(p_1^2+p_2^2)p_1p_2\mathbb{E}[\cos\theta_{12}] + 4p_1^2p_2^2\mathbb{E}[\cos^2\theta_{12}].
\end{aligned} \tag{3.8}$$

We assume the best case scenario for an estimator with no θ_{12} dependence, i.e. that $\mathbb{E}[\cos\theta_{12}] = 0$. In other words, on average the dot products between dipole moments cancel out. It is unlikely that all constituent dipoles in every molecule are orthogonal. For that reason, we assume that $\mathbb{E}[\cos^2\theta_{12}] > 0$, which is true as long as $\theta_{12} \neq n\frac{3\pi}{2}$ with $n \in \{x : x \in \mathbb{N}, x \neq 0\}$. Considering these assumptions, let our estimator predict the best possible (non-complex) estimation given its restrictions, $f = (p_1^2 + p_2^2)$, then we see that

$$\begin{aligned}
\mathbb{E}[(f-p^2)^2] &= f^2 - 2f(p_1^2+p_2^2) + (p_1^2+p_2^2)^2 + 4p_1^2p_2^2\mathbb{E}[\cos^2\theta_{12}] \\
&= 4p_1^2p_2^2\mathbb{E}[\cos^2\theta_{12}] > 0.
\end{aligned} \tag{3.9}$$

Therefore, it is shown that any estimator which is θ_{12} independent is biased when predicting the squared dipole moment magnitude, i.e. there exists irreducible error in the prediction of the expectation value of the squared loss.

Since L0Net and SchNet only have access to pairwise distances and only calculate scalars, talking about the addition of dipoles is nonsensical. These networks can only do calculations which involve the addition of magnitudes of dipoles. From the argumentation above, that means they are biased estimators for predicting the magnitude of the dipole moment.

Another way to see it is that L1Net allows vectors to interact by including the dot product operation within the Clebsch-Gordon decomposition, i.e. when two vector features are combined into a scalar feature using the Clebsch-Gordon decomposition their dot product is calculated. Since the Clebsch-Gordon coefficients are all equal to one for scalar valued inputs, L0Net does not include the dot product and cannot accurately interact vectors, which is necessary for predicting the magnitude of the dipole moment without bias.

The effects of this are seen empirically due to the significant increase in accuracy when including the first order spherical harmonic component in Figure 3.3 as well as in the state of the art prediction of the dipole moment in Table 3.5.

Why did e3nn perform well on this task compared with Cormorant? The major difference between e3nn and Cormorant is the inclusion of the Gated Block. (As well as the lack of the so-called two-body term.) One of the core reasons neural networks are universal function approximators is because of their nonlinearities. Cormorant claims that the Clebsch-Gordon transformation is nonlinear “enough” that they do not need to include other nonlinearities; however, the performance on this task is empirical evidence that e3nn is generalizing better. Unlike below with the isotropic polarizability, Cormorant used a similar batch size of 25 molecules per batch. Without another obvious culprit, the author hypothesizes that the Gated Block is necessary for accurate predictions on QM9 and especially on the dipole moment.

3.3.2 Isotropic Polarizability

The isotropic polarizability of a molecule is the constant which couples an applied electric field **E** to the induced dipole **p** by the following equation,

$$\mathbf{p} = \alpha \mathbf{E}. \tag{3.10}$$

The isotropic polarizability is merely an approximation of a higher-order tensor that couples an applied electric field to its induced dipole $\mathbf{p}$ using this equation,

$$p_i = \sum_j \alpha_{ij} E_j. \tag{3.11}$$

The approximation (3.10) assumes that polarizability is a linear effect, i.e. applied electric fields induce a dipole which is proportional to the electric field. In reality this is not the whole story and there are anti-isotropic effects; however, that information is not contained in the QM9 data set so we cannot attempt to model it. It is a shame because this quantity would be a perfect tensor to approximate with an L2Net. L2Nets have features which share the rotation order of the polarizability tensor itself, therefore an L2Net would probably predict those quantities quite accurately.

Since the network performed so well predicting the isotropic polarizability α, it is worth figuring out why. Unlike with the dipole moment μ, L0Net and L1Net have approximately the same generalization error on this quantity. It's very possible that L1Net included necessary rotations like with μ; however, we presume that these effects are small and offer another explanation.

e3nn has been trained on very small batch sizes compared to SchNet which was trained on batch sizes of 100 molecules per batch. As discussed in Section 3.1.3, we can validate that batch size played a role by training the a SchNet network with smaller batch sizes. This is easily done by modifying the tutorial example in schnetpack [41]. In Figure 3.8, one can see the training progress of a basic SchNet model against the isotropic polarizability target. Although the test set was not evaluated, the mean average validation error reduced by nearly an order of magnitude to rival the predictions of ARandNet. The validation mean average error after training turned out to be 0.0130.

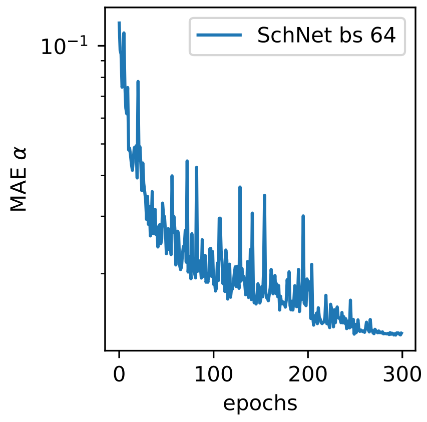

Figure 3.8: Mean average error in isotropic polarizability is plotted against epochs. The batch size was set to 64 molecules per batch. The training was accomplished using schnetpack with the default settings as found in `qm9_tutorial.py`. Using a small batch size significantly reduced the validation mean average error from schnetpack's reported 0.124 test error down to 0.0130 validation error.

3.3.3 Other Targets

The performance on other targets was varied. Notably, the electronic spatial extent, $\langle R^2 \rangle$, was also a state of the art prediction. However, it was difficult to find information about what this quantity really means. The computational chemistry tool Gaussian [45] reports it. It seems like this was the only reason why it is included in QM9. Given the lack of information all we can do is celebrate the high accuracy of the prediction.

The core target of many previous papers is the energy at absolute zero U_0. Although e3nn did not outperform other networks on this target, it is worth nothing that L1Net performed better than L0Net. It is plausible that including higher rotation orders in the calculation of energy has only a perturbative effect for most molecules in QM9. The same might be the case for other targets but more study is required. In particular, it would be interesting to calculate the energy of a system which contained a large number of interacting dipoles and then compare the energy prediction of L0Net and L1Net. On such a data set, we predict that L0Net and SchNet would perform poorly while L1Net would more accurately predict the energy.

Chapter 4

Conclusion

This document presented the results of the benchmark performance of e3nn on the QM9 data set as well as the necessary background for said task. The theory section explained how to train machine learning algorithms in general along with a break down of their error into bias, variance, and irreducible error. The methods which produced the QM9 data set itself were presented, namely density functional theory. The background for building a neural network equivariant to the special euclidean (3) group was presented along with the specifics of how e3nn works. The author contributed to e3nn in several ways during this project, rising to the third place in number of contributions to the software. Those contributions can be enumerated specifically on GitHub [17]. The most relevant addition was the introduction of the KernelConv module which reduced the memory footprint by slightly less than half and increased the speed by nearly two times. This module made doing the experiment on QM9 possible without prohibitively long wait times.

The experiment section included a comparison of other architectures to e3nn on different targets in QM9. The most important advantage of e3nn is in the accuracy of predictions on the dipole moment. Other hyperparameters were compared systematically including batch size and radial basis. The random hyperparameter search uncovered an architecture of e3nn which had state of the art performance on three of twelve QM9 targets, dipole moment, isotropic polarizability, and electronic spatial extent. The performance improvement on the dipole moment was very clearly the result of including the first order spherical harmonic features within the network. The isotropic polarizability accuracy was mostly attributable to the use of small batch sizes for training. Since electronic spatial extent was not clearly defined, it is not obvious why L1Net performed so well on that target.

The final messages of this work are: When predicting geometric tensors, consider utilizing features which have a similar rotation order to the predicted tensor. This applies even when calculating magnitudes as seen in the dipole moment. Gated nonlinearities have shown their value empirically; they should be used when possible. Considering the batch size has a strong effect in networks which operate on point clouds like these ones: Choose the hyperparameter carefully, erring on the side of smaller batches. Applying these techniques led to state of the art performance on the QM9 data set.

Appendix A

Definitions of Common Operations

The expectation value $\mathbb{E} : (\Omega \to \mathbb{R}) \to \mathbb{R}$ of a random variable $U : \Omega \to \mathbb{R}$ with sample space Ω and probability distribution p_U is defined as

$$\mathbb{E}[U] = \int_{\mathbb{R}^N} u \cdot p_U(u)\, du. \tag{A.1}$$

The variance of var of U is defined

$$\operatorname{var}[U] = \int_{\mathbb{R}^N} (u - \mathbb{E}[U]) \cdot (u - \mathbb{E}[U])^\top \cdot p_U(u)\, du. \tag{A.2}$$

The rules of probability for random variable X, Y with corresponding probability distributions $p(X), p(Y)$ are as follows. The sum rule is given as

$$p(X) = \sum_Y p(X, Y) \tag{A.3}$$

where $p(X, Y)$ is called the joint probability distribution and means the probability of both X and Y. The product rule of probability is given as

$$p(X, Y) = p(Y|X) \tag{A.4}$$

where $p(Y|X)$ is called the conditional probability distribution and it means the probability of Y given that X has occurred. Together these rules are commonly applied to write a relationship between conditional probabilities and called Bayes' Theorem

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} = \frac{p(X|Y)p(Y)}{\sum_Y p(X|Y)p(Y)}. \tag{A.5}$$

Appendix B

Kernel Normalization Constants

The author acknowledges the help of Mario Geiger with these calculations; a calculation of the normalization constants follows. We use the notation $\langle H \rangle$ for the mean of H. We first write four useful statements:

$$\text{(by independence)}\ \ \mathrm{var}\left[\sum_{l_{in}\,vj} K^{l_{out}l_{in}}_{ui\,vj} F^{l_{in}}_{vj}\right] = \sum_{l_{in}\,vj} \mathrm{var}\left[K^{l_{out}l_{in}}_{ui\,vj} F^{l_{in}}_{vj}\right], \tag{B.1}$$

$$\text{(by independence)}\ \ \mathrm{var}\left[K^{l_{out}l_{in}}_{ui\,vj} F^{l_{in}}_{vj}\right] = \langle (KF)^2 \rangle - \langle KF \rangle^2 = \langle K^2 \rangle \langle F^2 \rangle - \langle K \rangle^2 \langle F \rangle^2, \tag{B.2}$$

$$\text{(since } \langle R \rangle = 0\text{)}\ \ \langle K^{l_{out}l_{in}}_{ui\,vj} \rangle = \sum_{l_f\,k} \langle C^{l_{out}l_{in}l_f}_{ijk}\, Y^{l_f}_k\, R^{l_{out}l_{in}l_f}_{uv}\, n^{l_{out}l_{in}} \rangle = 0, \tag{B.3}$$

$$\begin{aligned} \langle (K^{l_{out}l_{in}}_{ui\,vj})^2 \rangle &= \sum_{l_f\,k} \sum_{l'_f\,k'} \langle C^{l_{out}l_{in}l_f}_{ijk}\, C^{l_{out}l_{in}l'_f}_{ijk'}\, Y^{l_f}_k\, Y^{l'_f}_{k'}\, R^{l_{out}l_{in}l_f}_{uv}\, R^{l_{out}l_{in}l'_f}_{uv}\, (n^{l_{out}l_{in}})^2 \rangle \\ \text{(since } \langle RR \rangle = \delta\text{)} &= \sum_{l_f} \sum_{kk'} C^{l_{out}l_{in}l_f}_{ijk}\, C^{l_{out}l_{in}l_f}_{ijk'}\, Y^{l_f}_k\, Y^{l_f}_{k'}\, (n^{l_{out}l_{in}})^2 \\ &= (n^{l_{out}l_{in}})^2 \sum_{l_f} \left(\sum_k C^{l_{out}l_{in}l_f}_{ijk}\, Y^{l_f}_k \right)^2. \end{aligned} \tag{B.4}$$

Now we can combine (B.3) and (B.4) with (B.2) to write,

$$\mathrm{var}\left[K^{l_{out}l_{in}}_{ui\,vj} F^{l_{in}}_{vj}\right] = (n^{l_{out}l_{in}})^2 \langle (F^{l_{in}}_{vj})^2 \rangle \sum_{l_f} \left(\sum_k C^{l_{out}l_{in}l_f}_{ijk}\, Y^{l_f}_k \right)^2, \tag{B.5}$$

which means that (B.1) can be written

$$\begin{aligned}
\text{(B.1)} &= \sum_{l_{in}} (n^{l_{out} l_{in}})^2 \sum_{vj} \tau_{l_{in}}^2 \sum_{l_f} \left(\sum_k C_{ijk}^{l_{out} l_{in} l_f} \, Y_k^{l_f} \right)^2 \\
&= \sum_{l_{in}} \left(\sum_v 1 \right) (n^{l_{out} l_{in}})^2 \tau_{l_{in}}^2 \sum_{l_f} \sum_j \left(\sum_k C_{ijk}^{l_{out} l_{in} l_f} \, Y_k^{l_f} \right)^2 \\
&= \sum_{l_{in}} \left(\sum_v 1 \right) (n^{l_{out} l_{in}})^2 \tau_{l_{in}}^2 \sum_{l_f} (4\pi(2l_{out}+1))^{-1} \\
&= (4\pi(2l_{out}+1))^{-1} \sum_{l_{in}} \left(\sum_v 1 \right) (n^{l_{out} l_{in}})^2 \, \tau_{l_{in}}^2 \left(\sum_{l_f} 1 \right),
\end{aligned} \tag{B.6}$$

where $\langle (F_{vj}^{l_{in}})^2 \rangle \coloneqq \tau_{l_{in}}^2$. Note that we want (B.1) $= \tau_{l_{in}}^2$, and we assume that $\tau_{l_{in}}^2 = 1$. Therefore,

$$\begin{aligned}
4\pi(2l_{out}+1) &= \sum_{l_{in}} \left(\sum_v 1 \right) (n^{l_{out} l_{in}})^2 \left(\sum_{l_f} 1 \right) \\
(n^{l_{out} l_{in}})^2 &= \frac{4\pi(2l_{out}+1)}{\left(\sum_v 1\right) \left(\sum_{l_f} 1\right) \left(\sum_{l_{in}} 1\right)}.
\end{aligned} \tag{B.7}$$

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[3] Gideon Lewis-Kraus. The great ai awakening. *The New York Times*, Dec 2016.

[4] From not working to neural networking. *The Economist*, June 2016.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[7] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147, 2019.

[8] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[9] Frank Noé, Alexandre Tkatchenko, Klaus-Robert Müller, and Cecilia Clementi. Machine learning for molecular simulation. *arXiv preprint arXiv:1911.02792*, 2019.

[10] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 2019.

[11] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.

[12] Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 52(11):2864–2875, 2012.

[13] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chem. Sci.*, 9:513–530, 2018.

[14] Jonathan M. Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M. Donghia, Craig R. MacNair, Shawn French, Lindsey A. Carfrae, Zohar Bloom-Ackerman, Victoria M. Tran, Anush Chiappino-Pepe, Ahmed H. Badran, Ian W. Andrews, Emma J. Chory, George M. Church, Eric D. Brown, Tommi S. Jaakkola, Regina Barzilay, and James J. Collins. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702.e13, Feb 2020.

[15] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, Andrew Palmer, Volker Settels, Tommi Jaakkola, Klavs Jensen, and Regina Barzilay. Correction to analyzing learned molecular representations for property prediction. *Journal of Chemical Information and Modeling*, 59(12):5304–5305, 2019. PMID: 31814400.

[16] K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller. Schnet – a deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24):241722, 2018.

[17] Mario Geiger, Tess Smidt, Benjamin K. Miller, Wouter Boomsma, Kostiantyn Lapchevskyi, Maurice Weiler, Michał Tyszkiewicz, and Jes Frellsen. github.com/e3nn/e3nn, March 2020.

[18] Brandon Anderson, Truong Son Hy, and Risi Kondor. Cormorant: Covariant molecular neural networks. In *Advances in Neural Information Processing Systems*, pages 14510–14519, 2019.

[19] Justin S Smith, Olexandr Isayev, and Adrian E Roitberg. Ani-1: an extensible neural network potential with dft accuracy at force field computational cost. *Chemical science*, 8(4):3192–3203, 2017.

[20] Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. In *International Conference on Learning Representations (ICLR)*, 2020.

[21] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[22] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[24] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[25] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization, 2016.

[26] Claude Sammut and Geoffrey I. Webb, editors. *Bias Variance Decomposition*. Springer US, Boston, MA, 2017.

[27] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[28] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[31] Larry A Curtiss, Paul C Redfern, and Krishnan Raghavachari. Gaussian-4 theory using reduced order perturbation theory. *The Journal of chemical physics*, 127(12):124105, 2007.

[32] Wolfram Koch and Max C Holthausen. *A chemist's guide to density functional theory*. John Wiley & Sons, 2015.

[33] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data, 2018.

[34] Howard Georgi. *Lie Algebras in Particle Physics: From Isospin to Unified Theories*. Westview Press, 1999.

[35] Taco Cohen, Mario Geiger, Jonas Köhler, Pim de Haan, K. T. Schütt, and Benjamin K. Miller. Lie learn. `https://github.com/AMLab-Amsterdam/lie_learn/releases/tag/v1.0_b`, 2020.

[36] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[37] Wikipedia contributors. Spherical harmonics, 2020. [Online; accessed 12-March-2020].

[38] Wikipedia contributors. Euler angles, 2020. [Online; accessed 27-March-2020].

[39] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5028–5037, 2017.

[40] Nathaniel Thomas, Tess Smidt, Steven M. Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

[41] K. T. Schütt, P. Kessel, M. Gastegger, K. A. Nicoli, A. Tkatchenko, and K.-R. Müller. Schnetpack: A deep learning toolbox for atomistic systems. *Journal of Chemical Theory and Computation*, 15(1):448–455, 2019.

[42] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks, 2018.

[43] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

[44] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[45] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, and D. J. Fox. Gaussian˜16 Revision C.01, 2016. Gaussian Inc. Wallingford CT.