Geometric Deep Learning:

A New Paradigm of Machine Learning

Amir P. Shanehsazzadeh

Advisor: Prof. Boaz Barak (Department of Computer Science)
Shadow Advisor: Prof. Clifford H. Taubes

Submitted in partial fulfillment of the honors requirements
for the degree of Bachelor of Arts in Mathematics

Department of Mathematics
Harvard University
March 21, 2022

Table of contents

Abstract

Geometric deep learning (GDL) uses the mathematical concepts of geometry to extend the powerful methods of deep learning to model data with complicated underlying manifolds such as molecules, proteins, social networks, and 3-dimensional images. More generally, it is a mathematical framework that unifies deep learning while also providing a method and theory to design high-quality models [1]. GDL proposes thinking of neural network architectures through the lens of invariants and symmetries. When thought of this way, we can compare GDL to Felix Wang's *Erlangen program* [2], which proposed methods to characterize different geometries. In this thesis, we present a self-contained presentation of the current state of geometric deep learning, both in terms of theory and practice, as presented in the seminal work by Bronstein et al. and in today's literature. Readers should have some background in statistics, linear algebra, and multivariable calculus. A background in algebra, geometry, and analysis is helpful for certain parts of the thesis, but we provide nearly all necessary definitions. In particular, we do not assume background in machine learning or deep learning. To that end, there may be content that seems rudimentary to either the machine learning expert or to the geometry expert, but this is necessary to enable accessibility for both groups. We begin by covering the fundamentals of machine learning and deep learning as well by providing motivation for GDL. We then develop the theory of GDL and utilize it to view existing deep learning models as GDL models and to propose a methodology for constructing GDL models. We conclude by presenting several applications of GDL. The contents of this thesis are of value to both practitioners and theorists interested in either geometry or machine learning.

Acknowledgements

I would like to thank Professor Boaz Barak and Professor Clifford H. Taubes for advising my thesis as well as my friends for their support throughout my undergraduate career. In particular, I want to thank Willam Zhang, Yash Nair, Ted Pyne, Junu Lee, Brian Tobin, and Abishrant Panday (who all supported my thesis writing to some extent and more generally my mathematics career) as well as Ilkin Bayramli, James Devaney, Alberto Mosconi, Diamante Balcazar, Henry Cerbone, Andrés Rosales, Eryk Pecyna, and Rohil Badkundri who have otherwise had a substantial impact on my college experience. It's been a tremendous ride gentlemen.

My journey to machine learning began in high school where I was fortunate to be advised by Dr. Roland Dunbrack. I still reminisce over the almost hour-long drives to his lab where we would discuss clustering algorithms and protein structure. I had the fortune to continue my foray into machine learning for biology at D. E. Shaw Research where I was advised by Dr. Paul Maragakis and Hunter Nisonoff. This was where I first worked on deep learning. I also had the privilege of working with the Sequin team at Google Brain where I was advised by Dr. David Belanger and David Dohan. Finally, the people who have had the greatest impact on my career to date are at Dyno Therapeutics. Specifically, I would like to thank Dr. Saum Sinai who first introduced me to Dyno and took a chance on me as well as Stephen Malina who has been a phenomenal manager and Eryney Marrogi who helped make me feel welcome. Seeing Dyno grow from a team that could (and would) fit in an over-sized cubical to nearly 100 people has been awe-inspiring.

Finally, I would like to thank my parents, Dr. Firoozeh Ghadyary-Khorzooghy and Mohammad Reza Shanehsazzadeh, who took the greatest chance on me by immigrating to the U.S. from Iran. Mom and dad, thanks for always being there for me. I owe you the world and, at the very least, this thesis which I dedicate to you.

Chapter 1

Introduction

There has never been a better time for the study and practice of machine learning [3, 4] than now. Machine learning, in particular deep learning [5], has in recent years revolutionized a variety of fields including natural language processing [6], computer vision [7], game theory [8], biology [9], chemistry [10], physics [11], and even pure mathematics [12]. This has resulted in an explosion of excitement for machine learning amongst academic groups, large companies such as Google and Meta which boast massive research divisions, and industry in general which has seen a substantial uptick in machine learning and deep learning startups across several sectors [13].

The topic of this thesis is not machine learning in general or even deep learning, but the emerging sub-field that is *geometric deep learning* (GDL) [1]. What exactly is geometric deep learning and how distinct is it from deep learning? At a high level, GDL is deep learning with the integration of geometric priors on the data. Let's motivate this with two examples, one toy and another practical.

Suppose that we are dealing with data that has the plus sign: $+$ as its underlying manifold (Definition 31). In other words, we assume that the data lies on a structure shaped liked a plus sign. Generally speaking, we assume a Euclidean geometry prior on our data, however it is clear that this geometry would not apply to our data well. To see why not, note that under the Euclidean prior we would assume that the distance between two adjacent ends of the plus sign, denoted as y, is only $\sqrt{2}$ times larger than the distance between an end to the center, denoted as x. In reality, the distance between two adjacent ends is 2 times larger than the distance between an end to the center (i.e. $y = 2x$), since to navigate a path between adjacent ends we would have to go through the center. The Euclidean prior assumes that straight-line navigation between points is possible, which is not always the case.

For a more practical example we will consider AlphaFold [9] from DeepMind[1]. AlphaFold is a highly accurate protein structure/folding prediction model that substantially outper-

[1]https://deepmind.com/

formed its competitors in the CASP14[2] structure prediction competition. Proteins consist of chains of amino acids which form a 3-D structure [14]. Proteins are equivariant under rigid motions, also known as Euclidean transformations, consisting of rotations and translations. To see this consider Figure 1.1 ("Ramachandran Plot" - Wikipedia), which depicts an idealized protein backbone along with the dihedral angles that define its structure. Rotating or translating the entirety of this structure does not alter it and so we expect a protein and a rotated and translated version of said protein to have the same properties. More mathe-

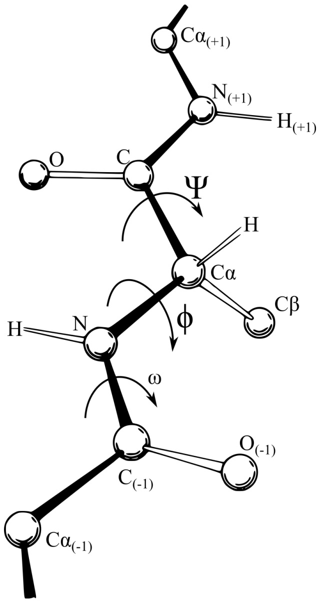


Figure 1.1: Idealized Protein Backbone with Dihedral angles Φ, Ψ, ω

matically, we say that proteins are SE(3)-equivariant where SE(3) is the special Euclidean group in 3-dimensions:

$$\mathrm{SE}(3) = \left\{ \boldsymbol{M} : \boldsymbol{M} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{r} \\ \boldsymbol{0}_{1\times 3} & 1 \end{pmatrix},\ \boldsymbol{r} \in \mathbb{R}^3,\ \boldsymbol{R} \in \mathbb{R}^{3\times 3},\ \boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{R}^\top \boldsymbol{R} = \mathbb{1},\ \det \boldsymbol{R} = 1 \right\}$$

AlphaFold's loss functions and novel attention mechanisms are designed to enable SE(3)-equivariance, allowing for the model to learn over the entire SE(3) group of transformed proteins, which is pivotal to its success [15]. We will discuss loss functions and the attention mechanism [16] later. At this point we only note their existence and importance.

The remainder of this introductory chapter presents the basics of machine learning and deep learning and discusses key challenges that motivate the field of geometric deep learning. We present this content in order to keep the thesis relatively self-contained and because machine learning is generally regarded as a statistics and computer science discipline and not a sub-field of mathematics. The more experienced reader is welcome and encouraged to skim or skip parts as they deem fit.

[2]https://predictioncenter.org/casp14/

1.1 Fundamentals of Machine Learning

What does it mean to learn? This question is beyond the scope of this thesis but the field of machine learning offers a small approximation to an answer for this question. Broadly speaking, a machine learning algorithm is an algorithm that is able to update itself or *learn* according to data it is provided [3]. We can further divide machine learning into three sub-disciplines: *supervised learning* which involves labeled data, *unsupervised learning* which involves unlabeled data, and *reinforcement learning* which involves agents learning how to maximize rewards. There are also more specific sub-disciplines such as *transfer learning* and *semi-supervised learning* where models first learn to tackle an “upstream” task and then attempt to use that knowledge to improve performance on a “downstream” task. We will explore each of these three key types of machine learning along with some examples.

1.1.1 Supervised Learning

In the supervised learning context our dataset $\mathcal{D}$ has labels:

$$\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}.$$

Each $\boldsymbol{x}_i$ is a set of features used to model the data and the corresponding value y_i is a label to be modeled. Features can be quantitative (such as age, height, or weight), qualitative (such as a text passage), and even high-dimensional or multi-dimensional (such as the pixels in an image or pairwise atomic distances in a molecule). Regardless of what the features are, we can model them as a high-dimensional vector $\boldsymbol{x}_i \in \mathbb{R}^n$. Labels can be just as complex as features but tend to be lower-dimensional, so we again model $y_i \in \mathbb{R}^m$. We generally assume that the elements of the dataset are sampled independently and with identical distribution (IID) from some distribution.

The goal of supervised learning is to find a model f that predicts the labels:

$$f : \mathbb{R}^n \to \mathbb{R}^m \text{ such that } \hat{y}_i = f(\boldsymbol{x}_i) \approx y_i.$$

To measure the performance of our model we use a *train set* and a *test set*, which are disjoint subsets of the original dataset:

$$\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}, \quad \mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset.$$

The reason for the disjointness is to measure the model’s performance on out-of-distribution data. Generally speaking, we randomly select a subset of the data (around 80% in practice) to be training data and the remainder to be test data. To train the model we optimize a *loss function* $\mathcal{L}$ of the model on the train data. At a high-level, $\mathcal{L}$ measures the deviation between the true labels y_i and the predicted labels $\hat{y}_i = f(\boldsymbol{x}_i)$ for $(\boldsymbol{x}_i, y_i) \in \mathcal{D}_{\text{train}}$. For the $m = 1$ case where our labels are scalars a common loss function is the *mean square error* or MSE:

$$\mathcal{L}_{\text{MSE}}(f) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_{\text{train}}} (y_i - \hat{y}_i)^2, \text{ where } \hat{y}_i = f(\boldsymbol{x}_i).$$

1.1.1.1 Predicting Protein Stability

Let's look at an example of supervised learning from the protein space. Modeling protein function is a problem of great interest [17]. In Shanehsazzadeh et al., the authors use *linear regression* and *convolutional neural network ensembles* to model protein stability, each of which are scalar values (we will explore these models later, for now consider them abstractly as functions). Each protein sequence is a length L string on an alphabet $\mathcal{A}$ of size approximately 20 (corresponding to the 20 amino acids): $|\mathcal{A}| = 20$. To represent each protein, we use the one-hot encoding where we represent the element $a_i \in \mathcal{A}$ with the vector e_i consisting of all zeroes except in the i^{th} index. We then map

$$\mathcal{A}^L \ni P = (a_{i_1}, a_{i_2}, ..., a_{i_L}) \to \begin{pmatrix} e_{i_1} \\ e_{i_2} \\ \vdots \\ e_{i_L} \end{pmatrix} \in \mathbb{R}^{L \times |\mathcal{A}|} = \mathbb{R}^{L \times 20}.$$

The models are trained to predict the scalar protein stability values. See Figure 1.2 (Figure 2 of Shanehsazzadeh et al. [17]) for plots of their predicted values versus the true values:

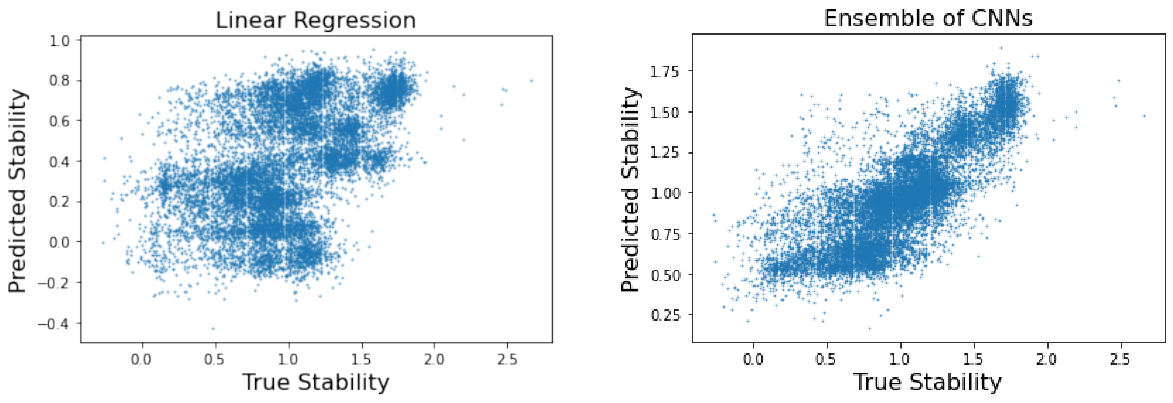


Figure 1.2: Predicted vs. True Stabilities for Linear Regression Model (Left) and Ensemble of CNN Models (Right). Note that an ensemble of models is an average of multiple models.

1.1.2 Unsupervised Learning

In the unsupervised learning context our dataset $\mathcal{D}$ does not have labels:

$$\mathcal{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}.$$

Each $\boldsymbol{x}_i$ is again a set of features like in the supervised learning context. The goal of unsupervised learning is to learn insights about the data without using labels. There are several examples of unsupervised learning but in many ways it is more like an art form than supervised learning is. Let's cover a few of these examples:

1.1.2.1 Clustering

Clustering involves grouping elements of a dataset $\mathcal{D}$ into disjoint clusters:

$$\mathcal{D} \to \{C_1, C_2, ..., C_n\} \text{ such that } \mathcal{D} = \bigcup_{i=1}^{n} C_i,\ C_i \subset \mathcal{D},\ C_i \cap C_j = \emptyset\ (i \neq j).$$

See Figure 1.3 ("Cluster analysis" - Wikipedia) for a visual of density-based clustering.

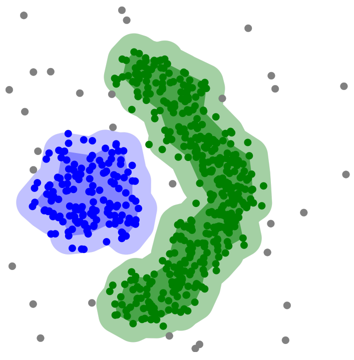


Figure 1.3: Density Based Clustering

Clustering is useful for knowledge discovery and has applications to many fields including bioinformatics (e.g. genetic sequence modeling) and marketing (e.g. determining groups of consumers based on shopping habits).

There are many different clustering algorithms and methods. We will look at two popular ones. The most popular algorithm is k-means, which is a centroid-based clustering algorithm that finds a set of k points $\boldsymbol{c} = \{\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_k\}$ (referred to as centroids) in the data and builds clusters $\boldsymbol{C} = \{C_1, C_2, ..., C_k\}$ around these points with $\boldsymbol{c}_i \in C_i$. The algorithm finds clusters $\boldsymbol{C}$ by minimizing the sum of the intra-cluster sum of squares:

$$\boldsymbol{C}^* = \underset{\boldsymbol{C}}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{c}_i\|^2,$$

where $\|\cdot\|$ indicates the 2-norm (Definition 18). Note that in general we will use this notation to indicate the 2-norm unless otherwise specified.

Another popular algorithm is density-based spatial clustering of applications with noise (DBSCAN) [18]. DBSCAN has two parameters $\varepsilon \in \mathbb{R}^+$ and minPts $\in \mathbb{N}$. The algorithm considers an ε-neighborhood (Definition 26) of each point in the data and any point that has at least minPts many neighbors (points in said neighborhood) is labeled a *core point*. Then points are connected to each other based on ε-proximity and the connected components of this graph are taken to create initial clusters. Any remaining points are assigned to a cluster if they are an ε-neighbor of (within ε of) any point in that cluster, otherwise they are labeled as *noise*. DBSCAN is useful when the number of clusters is difficult to predict and when the structure of the data is not well understood, but it is still sensitive to its parameters.

1.1.2.2 Language Modeling

Language modeling is a task in natural language processing that aims to learn a model f that predicts a probability distribution over sequences of words:

$$f(w_1, w_2, ..., w_n) \leftrightarrow \mathbb{P}(w_1, w_2, ..., w_n),\ \ w_i \text{ a word in the corresponding language},$$

where the bidirectional arrow indicates a correspondence between f and $\mathbb{P}$ (most models do not predict the probability directly). More intuitively, note that the sentence *"I love my mother"* should have a much higher probability than the string of words *"mother boat dog Jeep."* This is an unsupervised learning task because the model sees unlabeled sequences of words in a particular language.

In the past, Markov models were used for language modeling, however deep learning, specifically the transformer model [16], is the state-of-the-art today. A major moment for the field occurred when OpenAI[3] unveiled Generative Pre-trained Transformer 3 (GPT-3) [6], a 175 billion parameter deep language model that shattered previous state-of-the-art results in multiple language tasks including question-answering, reading comprehension, translation, text generation, and even arithmetic. GPT-3 is perhaps the closest model to passing a language-based Turing test [19].

1.1.3 Reinforcement Learning

Reinforcement learning describes algorithms designed to have an agent learn an optimal policy to maximize the reward it receives [20]. The reward is modeled by a *reward function*, which could range from binary functions (such as winning or losing a game) to more complicated functions (such as total winnings in a game of poker). The standard model for reinforcement learning uses a Markov decision process (MDP) model which consists of:

1. S a set of states for the environment and agent
2. A a set of actions the agent can take
3. $P(s, a, s') = \mathbb{P}(S_{t+1} = s'|S_t = s, A_t = 0)$, which is the transition probability of the agent ending up at state s' at time $t+1$ having been at state s and taking action a at time t
4. $R(s, a, s')$, which is the reward the agent receives after moving from state s to s' via action a

The goal of reinforcement learning is to learn an optimal policy π defined as

$$\pi : S \times A \to [0, 1]\ \text{ such that }\ \pi(s, a) = \mathbb{P}(A_t = a|S_t = s),$$

[3]https://openai.com/

where optimality means that π maximizes cumulative rewards.

A variety of methods exist to solve for π including value iteration, policy iteration, temporal difference learning, Q-learning, and deep learning methods such as deep Q-learning [21].

Reinforcement learning has been highly successful at learning to play games at superhuman levels. These games can be of quite high complexity, even including the game of Go. In fact, DeepMind's AlphaGo model [8] was able to beat the world Go champion Lee Sedol[4]. More recently, DeepMind's MuZero model [22] was able to master a number of games without even knowing the rules and it was recently used for YouTube video compression[5], opening the door to industrial applications.

1.1.4 Classical Machine Learning

While the topic of this thesis is a sub-field of deep learning, it is important to understand classical machine learning which preceded deep learning. Classical machine learning models are generally much less complex and "shallow" than deep learning models. They tend to be more interpretable as well and require feature engineering. We dive into the tried and true linear regression model as an example.

1.1.4.1 Linear Regression

Linear regression is used for supervised learning. For a practical example see Figure 1.2. Suppose we have n-dimensional features and want to model scalar labels. Then a linear regression model is parametrized by $\boldsymbol{\beta} \in \mathbb{R}^{n+1}$ such that:

$$\hat{y}_i = f_{\boldsymbol{\beta}}(\boldsymbol{x}_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in} = \beta_0 + \sum_{j=1}^{n} \beta_j x_{ij}.$$

If we let $\boldsymbol{x}'_i = (1, x_{i1}, x_{i2}, ..., x_{in})$ and have N samples in our dataset then we can use matrix notation to write

$$\boldsymbol{y}' = \boldsymbol{X}\boldsymbol{\beta} \text{ where } \boldsymbol{y}' = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \ \boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}'_1 \\ \boldsymbol{x}'_2 \\ \vdots \\ \boldsymbol{x}'_N \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nn} \end{pmatrix}, \ \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}.$$

To solve for $\boldsymbol{\beta}$ we minimize the loss function $\mathcal{L}(f_{\boldsymbol{\beta}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f_{\boldsymbol{\beta}}(\mathbf{x}_i))^2$.

Claim 1. The optimal solution that minimizes the loss function $\mathcal{L}(f_{\boldsymbol{\beta}})$ is $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$ where $\boldsymbol{y} = \begin{pmatrix} y_1 & y_2 & \dots & y_n \end{pmatrix}^\top$.

[4]The Google DeepMind challenge match - DeepMind
[5]MuZero's first step from research into the real world - DeepMind Blog

Proof. Write

$$\mathcal{L}(f_{\boldsymbol{\beta}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f_{\boldsymbol{\beta}}(\mathbf{x}_i))^2 \propto \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\intercal(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$$

$$= \boldsymbol{y}^\intercal\boldsymbol{y} - \boldsymbol{y}^\intercal\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{\beta}^\intercal\boldsymbol{X}^\intercal\boldsymbol{y} + \boldsymbol{\beta}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{\beta}.$$

Then compute the gradient of the loss function

$$\frac{\partial\mathcal{L}(f_{\boldsymbol{\beta}})}{\partial\boldsymbol{\beta}} \propto -2\boldsymbol{X}^\intercal\boldsymbol{y} + 2\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{\beta}.$$

Setting the gradient to 0 gives us $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\intercal\boldsymbol{X})^{-1}\boldsymbol{X}^\intercal\boldsymbol{y}$. The last step is to show that the Hessian is positive definite. Note that $\frac{\partial^2\mathcal{L}(f_{\boldsymbol{\beta}})}{\partial\boldsymbol{\beta}^2} \propto 2\boldsymbol{X}^\intercal\boldsymbol{X}$. The matrix $\boldsymbol{X}^\intercal\boldsymbol{X}$ is a Gram matrix which is positive semi-definite and positive definite if and only if the columns of $\boldsymbol{X}$ are linearly independent, which we assume. □

We see that the solution to linear regression is quite tractable. It is also worth noting that the model is very interpretable. The intercept β_0 is the mean value assuming all features are 0. The weights β_i for $i \geq 1$ correspond to the i^{th} feature in the data. The sign of β_i gives the direction of the correlation and $|\beta_i|$ gives a magnitude of the impact of the i^{th} feature.

1.1.5 Deep Learning and Neural Networks

Our last stop before covering geometric deep learning is an overview of deep learning basics. In this thesis, we will consider deep learning models to be neural networks.

Neural networks consist of multiple layers of input-output transformations with non-linear activations in between. The standard non-linear activation function is the Rectified Linear Unit (ReLU):

$$\text{ReLU}(x) = \max(0, x).$$

The intuition behind the ReLU is that it mimics human neurons which do not fire unless they reach some activation threshold. Mathematically, non-linear activations give deep learning models their powerful expressivity. We thus write this model in the form

$$F_{\text{Deep}}(\mathbf{x}) = A_n \circ \text{ReLU} \circ A_{n-1} \circ \text{ReLU} \circ \cdots \circ A_2 \circ \text{ReLU} \circ A_1(\mathbf{x}),$$

where n is the number of layers, each A_i is a matrix transformation of appropriate dimension, and ReLU applied to a vector is done so element-wise. If each A_i is a linear map then we say the network is *fully-connected*. Generally speaking, for a model to be "deep" we take $n > 2$ (otherwise, it is considered a shallow model). Also, note that ReLU is not the only non-linear activation function and multiple others are used in practice.

Neural networks have massive expressive power, which is often described using a variety of *universal approximation theorems*. These theorems state something along the lines of "for any 'reasonable' function there exists a neural network with finite depth (number of layers) that can approximate it." We present one of these theorems and its weak converse here. The theorem is adapted from Park et al. and for a proof we recommend their paper to the reader [23].

Theorem 2 (Universal Approximation Theorem)**.** *Define a fully-connected ReLU network of depth d to be a neural network with d layers with ReLU activations between layers. For any $f \in L^p(\mathbb{R}^n, \mathbb{R}^m)$ and $\varepsilon > 0$ there exists a fully-connected ReLU network F of depth $d_{nm} = \max(n+1, m)$ such that*

$$\int_{\mathbb{R}^n} \|f(x) - F(x)\|^p < \varepsilon.$$

Additionally, there exists some $f \in L^p(\mathbb{R}^n, \mathbb{R}^m)$ and $\varepsilon > 0$ such that the above approximation bound does not hold for any fully-connected ReLU network F with depth less than d_{nm}.

With Theorem 2 we see that as long as our data can be modeled by a "reasonable" function (a function in L^p (Definition 19)) there exists some neural network that can approximate it arbitrarily closely. This implies the expressive power of neural networks, but we have not yet shown how to find a neural network to model a specific task. Unlike the case of classical machine learning models such as linear regression, neural networks can become very complicated both in terms of size (GPT-3 [6] has 175 billion parameters) and in terms of complexity (the non-linear activations, for example, make the function complicated). Because of this, we cannot hope to analytically optimize the loss function by computing its gradient and setting it to 0 (like we did for linear regression). However, the gradient of the loss function is still relevant to optimizing it.

Neural networks are trained using the gradient descent algorithm. The intuition behind gradient descent is that moving against the gradient of a differentiable function leads to a local minimum of said function. Gradient descent works as follows:

1. Let $F_{\boldsymbol{\omega}}$ be a neural network with parameters $\boldsymbol{\omega}$. Initialize $\boldsymbol{\omega}_0 = \boldsymbol{\omega}$ (note that $\boldsymbol{\omega}$ can be randomly initialized).

2. Consider the loss function $\mathcal{L}(F_{\boldsymbol{\omega}})$, which is a function of the parameters $\boldsymbol{\omega}$. The loss $\mathcal{L}$ could be a variety of things as long as it is a proxy for some kind of model performance and is differentiable, convex, and has Lipschitz continuous gradient. Two common loss functions are the mean squared error (MSE) for regression and the cross-entropy loss (CE) for classification. We define these loss functions here. Note that for the cross-entropy loss we expect the model to predict one of k categories and thus it should output a vector $(p_1, p_2, ..., p_k)$ with p_i the probability that the input is assigned to category i (as a result $p_i \in [0,1]$ and $\sum_{i=1}^{k} p_i = 1$).

$$\mathcal{L}_{\text{MSE}}(F_{\boldsymbol{\omega}}) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} (y - F_{\boldsymbol{\omega}}(\boldsymbol{x}))^2, \ y \in \mathbb{R}.$$

$$\mathcal{L}_{\text{CE}}(F_{\boldsymbol{\omega}}) = -\frac{1}{|\mathcal{D}|}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \log(F_{\boldsymbol{\omega}}(\boldsymbol{x})_y),\ y \in \{1, 2, ..., k\}.$$

3. Repeatedly update the model parameters using the formula:

$$\boldsymbol{\omega}_{n+1} = \boldsymbol{\omega}_n - \eta\nabla\mathcal{L}(F_{\boldsymbol{\omega}_n}),\ \eta \in \mathbb{R}^+.$$

   Continue updating the parameters until some condition is met, such as the loss attaining some value or after some number of iterations are completed. Note that the parameter η is referred to as the learning rate. It is generally small to prevent the update from overshooting the local minimum.

We now prove that gradient descent minimizes the loss function (under certain assumptions).

Theorem 3 (Convergence of Gradient Descent)**.** *Let $\mathcal{L}$ be a loss function. We abuse notation and write $\mathcal{L}(\boldsymbol{\omega})$ instead of $\mathcal{L}(F_{\boldsymbol{\omega}})$ to highlight the fact that the loss is a function of the parameters $\boldsymbol{\omega} \in \mathbb{R}^p$. We assume that $\mathcal{L}$ is convex, differentiable, and has an L-Lipschitz continuous gradient with respect to $\boldsymbol{\omega}$ so that $\|\nabla\mathcal{L}(\boldsymbol{\omega}) - \nabla\mathcal{L}(\boldsymbol{\omega}')\| \leq L\|\boldsymbol{\omega} - \boldsymbol{\omega}'\|$. Suppose that the optimal set of parameters are $\boldsymbol{\omega}^* = \operatorname{argmin}_{\boldsymbol{\omega}\in\mathbb{R}^k}\mathcal{L}(\boldsymbol{\omega})$. If we run k iterations of gradient descent with learning rate $\eta \leq \frac{1}{L}$ and starting at $\boldsymbol{\omega}_0$ then we will find $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, ...\boldsymbol{\omega}_k$ such that*

$$\mathcal{L}(\boldsymbol{\omega}_1) > \mathcal{L}(\boldsymbol{\omega}_2) > ... > \mathcal{L}(\boldsymbol{\omega}_k) \geq \mathcal{L}(\boldsymbol{\omega}^*),$$

and, in particular, gradient descent has convergence rate $O\left(\frac{1}{k}\right)$:

$$\mathcal{L}(\boldsymbol{\omega}_k) - \mathcal{L}(\boldsymbol{\omega}^*) \leq \frac{\|\boldsymbol{\omega}^* - \boldsymbol{\omega}_0\|^2}{2k\eta}.$$

Proof. We adapt a proof from [4, 24].

We take a 2nd order approximation of $\mathcal{L}$ around $\mathcal{L}(\boldsymbol{\omega})$. It follows from the fact that $\nabla\mathcal{L}$ is L-Lipschitz that

$$\mathcal{L}(\boldsymbol{\omega}') \leq \mathcal{L}(\boldsymbol{\omega}) + \nabla\mathcal{L}(\boldsymbol{\omega})^{\mathsf{T}}(\boldsymbol{\omega}' - \boldsymbol{\omega}) + \frac{1}{2}\nabla^2\mathcal{L}(\boldsymbol{\omega})\|\boldsymbol{\omega}' - \boldsymbol{\omega}\|^2$$

$$\leq \mathcal{L}(\boldsymbol{\omega}) + \nabla\mathcal{L}(\boldsymbol{\omega})^{\mathsf{T}}(\boldsymbol{\omega}' - \boldsymbol{\omega}) + \frac{1}{2}L\|\boldsymbol{\omega}' - \boldsymbol{\omega}\|^2.$$

Now substitute $\boldsymbol{\omega}_n$ for $\boldsymbol{\omega}$ and $\boldsymbol{\omega}_{n+1} = \boldsymbol{\omega}_n - \eta\nabla\mathcal{L}(\boldsymbol{\omega}_n)$ for $\boldsymbol{\omega}'$:

$$\mathcal{L}(\boldsymbol{\omega}_{n+1}) \leq \mathcal{L}(\boldsymbol{\omega}_n) + \nabla\mathcal{L}(\boldsymbol{\omega}_n)^{\mathsf{T}}(\boldsymbol{\omega}_{n+1} - \boldsymbol{\omega}_n) + \frac{1}{2}L\|\boldsymbol{\omega}_{n+1} - \boldsymbol{\omega}_n\|^2$$

$$= \mathcal{L}(\boldsymbol{\omega}_n) - \eta\nabla\mathcal{L}(\boldsymbol{\omega}_n)^{\mathsf{T}}\nabla\mathcal{L}(\boldsymbol{\omega}_n) + \frac{1}{2}L\|\eta\nabla\mathcal{L}(\boldsymbol{\omega}_n)\|^2$$

$$= \mathcal{L}(\boldsymbol{\omega}_n) - \eta\left(1 - \frac{\eta L}{2}\right)\|\nabla\mathcal{L}(\boldsymbol{\omega}_n)\|^2$$

$$\leq \mathcal{L}(\boldsymbol{\omega}_n) - \frac{\eta}{2}\|\nabla\mathcal{L}(\boldsymbol{\omega}_n)\|^2,$$

where in the last line we use the fact that $\eta \leq \frac{1}{L} \implies 1 - \frac{\eta L}{2} \geq \frac{1}{2}$. The result implies that as long as $\|\nabla\mathcal{L}(\boldsymbol{\omega}_n)\|^2 > 0$ (i.e. we have not reached the minimum) we will have $\mathcal{L}(\boldsymbol{\omega}_n) > \mathcal{L}(\boldsymbol{\omega}_{n+1})$, as desired. Note that we relied on the learning rate being small: $\eta \leq \frac{1}{L}$. If the learning rate were too large the proof would not hold and we might "overshoot" the minimum.

Now for the convergence rate result, note that since $\mathcal{L}$ is convex we have

$$\mathcal{L}(\boldsymbol{\omega}^*) \geq \mathcal{L}(\boldsymbol{\omega}) + \nabla\mathcal{L}(\boldsymbol{\omega})^{\mathsf{T}}(\boldsymbol{\omega}^* - \boldsymbol{\omega}) \implies \mathcal{L}(\boldsymbol{\omega}) \leq \mathcal{L}(\boldsymbol{\omega}^*) + \nabla\mathcal{L}(\boldsymbol{\omega})^{\mathsf{T}}(\boldsymbol{\omega} - \boldsymbol{\omega}^*).$$

Substituting $\boldsymbol{\omega}_n$ for $\boldsymbol{\omega}$ and using our earlier result we have

$$\mathcal{L}(\boldsymbol{\omega}_n) \leq \mathcal{L}(\boldsymbol{\omega}^*) + \nabla\mathcal{L}(\boldsymbol{\omega}_n)^{\mathsf{T}}(\boldsymbol{\omega}_n - \boldsymbol{\omega}^*) \implies$$

$$\mathcal{L}(\boldsymbol{\omega}_{n+1}) \leq \mathcal{L}(\boldsymbol{\omega}^*) + \nabla\mathcal{L}(\boldsymbol{\omega}_n)^{\mathsf{T}}(\boldsymbol{\omega}_n - \boldsymbol{\omega}^*) - \frac{\eta}{2}\|\nabla\mathcal{L}(\boldsymbol{\omega}_n)\|^2 \implies$$

$$\mathcal{L}(\boldsymbol{\omega}_{n+1}) - \mathcal{L}(\boldsymbol{\omega}^*) \leq \frac{1}{2\eta}\left(2\eta\nabla\mathcal{L}(\boldsymbol{\omega}_n)^{\mathsf{T}}(\boldsymbol{\omega}_n - \boldsymbol{\omega}^*) - \eta^2\|\nabla\mathcal{L}(\boldsymbol{\omega}_n)\|^2\right)$$

$$= \frac{1}{2\eta}\left(\|\boldsymbol{\omega}_n - \boldsymbol{\omega}^*\|^2 - \left(\|\boldsymbol{\omega}_n - \boldsymbol{\omega}^*\|^2 - 2\eta\nabla\mathcal{L}(\boldsymbol{\omega}_n)^{\mathsf{T}}(\boldsymbol{\omega}_n - \boldsymbol{\omega}^*) + \eta^2\|\nabla\mathcal{L}(\boldsymbol{\omega}_n)\|^2\right)\right)$$

$$= \frac{1}{2\eta}\left(\|\boldsymbol{\omega}_n - \boldsymbol{\omega}^*\|^2 - \|\boldsymbol{\omega}_n - \eta\nabla\mathcal{L}(\boldsymbol{\omega}_n) - \boldsymbol{\omega}^*\|^2\right)$$

$$= \frac{1}{2\eta}\left(\|\boldsymbol{\omega}_n - \boldsymbol{\omega}^*\|^2 - \|\boldsymbol{\omega}_{n+1} - \boldsymbol{\omega}^*\|^2\right).$$

Finally, we derive the desired result:

$$\mathcal{L}(\boldsymbol{\omega}_k) - \mathcal{L}(\boldsymbol{\omega}^*) \leq \frac{1}{k}\sum_{n=1}^{k}\left(\mathcal{L}(\boldsymbol{\omega}_n) - \mathcal{L}(\boldsymbol{\omega}^*)\right) \leq \frac{1}{k}\sum_{n=1}^{k}\frac{1}{2\eta}\left(\|\boldsymbol{\omega}_{n-1} - \boldsymbol{\omega}^*\|^2 - \|\boldsymbol{\omega}_n - \boldsymbol{\omega}^*\|^2\right)$$

$$= \frac{1}{2k\eta}\left(\|\boldsymbol{\omega}_0 - \boldsymbol{\omega}^*\|^2 - \|\boldsymbol{\omega}_k - \boldsymbol{\omega}^*\|^2\right) \leq \frac{\|\boldsymbol{\omega}_0 - \boldsymbol{\omega}^*\|^2}{2k\eta}.$$

$\square$

In practice, gradient descent can often not be utilized because of the size of the dataset (which can make the loss function computation intractable). Instead, *stochastic gradient descent* (SGD) is used. With SGD, we split our data into multiple batches and use each of those batches to estimate the loss function by running the gradient descent update one batch at a time. This sacrifices some accuracy in the form of estimation inaccuracy but allows for tractability. SGD has similar convergence guarantees as gradient descent, which for the sake of brevity we will not present but we refer the reader to [4] for a proof. There are numerous versions of SGD used in the field such as the Adam optimizer [25].

The last thing to point out is the actual gradient computation. This can be done using the *backpropagation algorithm* which uses dynamic programming along with the chain rule from calculus. We will not discuss backpropagation extensively since it is primarily a practical detail and refer the curious reader to [4] for proper treatment of the algorithm.

Having presented some basic theoretical results about neural networks we now discuss some prominent neural network architectures.

1.1.5.1 Feedforward Neural Networks

Feedforward neural networks consist of compositions of linear maps (known as layers) along with non-linear activations between layers. For an n-dimensional feature vector and m-dimensional target vector let $f_1, f_2, ..., f_k$ be linear maps such that $f_i(\boldsymbol{v}) = \boldsymbol{W}_i\boldsymbol{v} + \boldsymbol{b}_i$ with $\boldsymbol{W}_1 \in \mathbb{R}^{h_1 \times n}$ and $\boldsymbol{b}_1 \in \mathbb{R}^{h_1}$, $\boldsymbol{W}_i \in \mathbb{R}^{h_{i+1} \times h_i}$ and $\boldsymbol{b}_i \in \mathbb{R}^{h_{i+1}}$ for $1 < i < k$, and $\boldsymbol{W}_k \in \mathbb{R}^{m \times h_k}$ and $\boldsymbol{b}_k \in \mathbb{R}^m$. The matrices $\boldsymbol{W}_i$ and vectors $\boldsymbol{b}_i$ are referred to as *weights* and *biases*, respectively. The variables $h_i \in \mathbb{N}$ are referred to as hidden dimensions as they correspond to the size of the *hidden layers* of the model (intermediary states of the model used for computation). Let $\phi_1, \phi_2, ..., \phi_{k-1}$ be non-linear activation functions (such as ReLU). We define a feedforward neural network F with depth (number of layers) k for $\boldsymbol{x} \in \mathbb{R}^n$ such that

$$F(\boldsymbol{x}) = f_k \circ \phi_{k-1} \circ f_{k-1} \circ \phi_{k-2} \circ f_{k-2} \circ \cdots \circ \phi_2 \circ f_2 \circ \phi_1 \circ f_1(\boldsymbol{x}) \in \mathbb{R}^m.$$

Note that the non-linear activations are essential for the model's expressive power since $A_i \circ A_{i-1}$ is simply another linear map. Also, note that this is not the only feedforward neural network architecture. There are many other versions that are in some way distinct from this standard model.

1.1.5.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are used to model sequential data such as time-series data. RNNs correspondingly have sequential hidden states that update each other while also updating and being updated based on the input. Let $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_\tau, y_\tau)$ be our sequential features and labels. Let $\{\boldsymbol{h}_t : 1 \leq t \leq \tau\}$ be hidden states. Let $\{\boldsymbol{U}_t : 1 \leq t \leq \tau\}$ be weights corresponding to input to hidden connections. Likewise let $\{\boldsymbol{V}_t : 1 \leq t \leq \tau\}$ correspond to hidden to input connections and let $\{\boldsymbol{W}_t : 1 \leq t \leq \tau\}$ correspond to hidden to hidden connections. Let $\{\boldsymbol{b}_t : 1 \leq t \leq \tau\}$ and $\{\boldsymbol{c}_t : 1 \leq t \leq \tau\}$ be biases. The RNN performs updates from $t = 1$ to $t = \tau$ as follows:

1. $\boldsymbol{a}_t = \boldsymbol{W}_t\boldsymbol{h}_{t-1} \ + \ \boldsymbol{U}_t\boldsymbol{X}_t \ + \ \boldsymbol{b}_t$ (update on previous hidden state and current input)
2. $\boldsymbol{h}_t = \phi_t(\boldsymbol{a}_t)$ (apply non-linear activation) — We often use $\phi_i(x) = \tanh(x) = 1 - \frac{2}{1+e^{2x}}$.
3. $\boldsymbol{o}_t = \boldsymbol{V}_t\boldsymbol{h}_t + \boldsymbol{c}_t$ (set current output state based on current hidden state)

4. $\hat{y}_t = T(\boldsymbol{o}_t)$ (predict output based on output state) — The function T depends on what kind of output we wish to predict. For classification with $y_t \in \{1, 2, ..., k\}$ we could use

$$T(\boldsymbol{o}_t) = \text{softmax}(\boldsymbol{o}_t)$$

$$= \left(\frac{1}{\sum_{i=1}^{k} \exp(\boldsymbol{o}_{ti})}\right) \begin{pmatrix} \exp(\boldsymbol{o}_{t1}) & \exp(\boldsymbol{o}_{t2}) & \dots & \exp(\boldsymbol{o}_{tk}) \end{pmatrix} = \begin{pmatrix} p_{t1} & p_{t2} & \dots & p_{tk} \end{pmatrix}.$$

Note that we have implicitly assumed $\boldsymbol{o}_t$ to be k-dimensional. We see that each $p_{ti} \in [0, 1]$ and $\sum_{i=1}^{k} p_{ti} = 1$, giving us a probability distribution. We then model $p_{ti} = \mathbb{P}(y_t = i)$.

RNNs are trained with a variant of backprogation known as *backpropagation through time* which computes gradients based on the recurrence relations by unwinding an RNN's computational graph. As was the case with feedforward neural networks, there are multiple variants of RNNs. For example, there is the Long Short Term Memory (LSTM) network, which is designed to deal with the *vanishing gradient problem.* RNNs rely on past hidden states (information in the past) to compute gradients. Looking too far back though can lead to arbitrarily small gradients which end up "vanishing" (being approximately 0). LSTMs are designed to mitigate this. Another interesting example is the Bidirectional LSTM (BiLSTM) which updates states both based on the previous state and the subsequent state. This is useful for modeling non-directional sequential data like protein sequences [26].

1.1.5.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are designed for grid-like data and use convolution somewhere in their architecture. For a discrete input x and kernel w the 1-dimensional convolution is given by:

$$C(i) = (x * w)(i) = \sum_{j=-\infty}^{\infty} x(j)w(i-j).$$

Note that we use a sum instead of an integral as we are doing a discrete convolution. For a 2-dimensional input X and kernel W we have the 2-D convolution:

$$C(i, j) = (X * W)(i, j) = \sum_{k} \sum_{\ell} X(k, \ell)W(i-k, j-\ell).$$

See Figure 1.4 (Page 69, Figure 14 of Bronstein et al. [1]) for a visual of convolution. We can think of convolution as matrix multiplication if we use highly constrained *Toeplitz matrices* or *circulant matrices.* In practice, various other methods are used to minimize computational cost.

CNNs offer a number of benefits including *sparseness*, *parameter sharing*, and *translational equivariance.* By choosing a kernel that is smaller than our input we can sparsely learn relevant features from our input while also sharing parameters for different regions of the input, giving us the former two properties. Translation equivariance follows from properties of the convolution. Specifically, if we take $f : X \to X'$ where $X'(x, y) = X(x - m, y - m)$ then applying f to X and convolving is equivalent to convolving and then applying f to the convolved output.

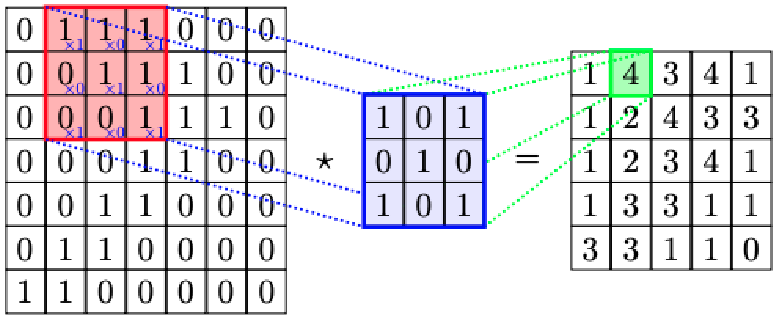


Figure 1.4: Convolution with a Filter (Blue)

After convolution there are usually two steps in a CNN. First, a non-linear activation function is applied to the convolved output. Then, we apply *pooling*, which is essentially an aggregation operation. For example, max pooling involves taking the largest value in each small grid of the convolved output. Operations like pooling can be thought of as coarsening their input. We will see the importance of coarsening in Chapter 2.

There are multiple different CNN architectures ranging from having just one convolution layer to having many. There are also different architectures such as *dilated convolutions* which "skip" parts of the input when performing convolution. For an example of CNNs in practice see Figure 1.2.

1.1.5.4 Transformers

The *transformer architecture* [16] and its *attention mechanism* have pushed the state-of-the-art in deep learning substantially further. The idea behind attention is to compute weights that represent sequential context and importance. Suppose we want to model a transformation between the sequences $\{\boldsymbol{x}_i : 1 \le i \le n\}$ and $\{\boldsymbol{y}_j : 1 \le j \le m\}$. How much should a particular $\boldsymbol{x}_i$ be weighed in determining a given $\boldsymbol{y}_j$? Attention mechanisms answer this question by determining an appropriate context weight w_{ij} for the impact of $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$ on each other.

For a more intuitive example, let's think about translation. Suppose I want to translate the English sentence *"I run every day"* to the Spanish sentence *"Corro cada día."* The word "run" is significant in determining the unconjugated Spanish verb "correr" and "I" determines the conjugation to "Corro" so "I run" has significant weight in determining "Corro." Similarly, "every day" translates directly to "cada día." We see then that "I run" should have significantly less weight than "every day" in translating to "cada día." Determining these weights is of great importance when dealing with language and as a result it should come as no surprise that transformer models such as GPT-3 [6] are the state-of-the-art in natural language processing.

We now present *scaled dot-product attention.* Our input consists of n query vectors $\{\boldsymbol{q}_i : 1 \leq i \leq n\}$ and n key vectors $\{\boldsymbol{k}_i : 1 \leq i \leq n\}$ both of dimension d_k as well as n value vectors $\{\boldsymbol{v}_i : 1 \leq i \leq n\}$ of dimension d_v. Define the matrices

$$\boldsymbol{Q} = \begin{pmatrix} — & \boldsymbol{q}_1 & — \\ — & \boldsymbol{q}_2 & — \\ & \vdots & \\ — & \boldsymbol{q}_n & — \end{pmatrix}, \ \boldsymbol{K} = \begin{pmatrix} — & \boldsymbol{k}_1 & — \\ — & \boldsymbol{k}_2 & — \\ & \vdots & \\ — & \boldsymbol{k}_n & — \end{pmatrix}, \ \boldsymbol{V} = \begin{pmatrix} — & \boldsymbol{v}_1 & — \\ — & \boldsymbol{v}_2 & — \\ & \vdots & \\ — & \boldsymbol{v}_n & — \end{pmatrix}.$$

Then the attention mechanism is defined as

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V},$$

where the softmax is applied row-wise. Let's break this down to better understand it. Note that the matrix product $\boldsymbol{Q}\boldsymbol{K}^\top$ consists of the dot products between the queries and the keys:

$$\boldsymbol{Q}\boldsymbol{K}^\top = \begin{pmatrix} — & \boldsymbol{q}_1 & — \\ — & \boldsymbol{q}_2 & — \\ & \vdots & \\ — & \boldsymbol{q}_n & — \end{pmatrix} \begin{pmatrix} | & | & & | \\ \boldsymbol{k}_1^\top & \boldsymbol{k}_2^\top & \dots & \boldsymbol{k}_n^\top \\ | & | & & | \end{pmatrix} = \left[\boldsymbol{q}_i \boldsymbol{k}_j^\top\right]_{1 \leq i,j \leq n}.$$

The row-wise softmax is determined by each query's dot products with the keys:

$$\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right) = \text{softmax}\left(\frac{\left[\boldsymbol{q}_i \boldsymbol{k}_j^\top\right]_{1 \leq i,j \leq n}}{\sqrt{d_k}}\right) = \begin{pmatrix} — & \boldsymbol{s}_1 & — \\ — & \boldsymbol{s}_2 & — \\ & \vdots & \\ — & \boldsymbol{s}_n & — \end{pmatrix}$$

$$\text{with } \boldsymbol{s}_i = \text{softmax}\left(\frac{1}{\sqrt{d_k}}\begin{pmatrix} \boldsymbol{q}_i\boldsymbol{k}_1^\top & \boldsymbol{q}_i\boldsymbol{k}_2^\top & \dots & \boldsymbol{q}_i\boldsymbol{k}_n^\top \end{pmatrix}\right).$$

We now have

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V} = \begin{pmatrix} — & \boldsymbol{s}_1 & — \\ — & \boldsymbol{s}_2 & — \\ & \vdots & \\ — & \boldsymbol{s}_n & — \end{pmatrix} \begin{pmatrix} — & \boldsymbol{v}_1 & — \\ — & \boldsymbol{v}_2 & — \\ & \vdots & \\ — & \boldsymbol{v}_n & — \end{pmatrix}.$$

We can think of each $\boldsymbol{s}_i$ as a weight vector corresponding to query i that uses a function of the dot product of $\boldsymbol{q}_i$ with $\boldsymbol{k}_j$ to determine the appropriate weight for $\boldsymbol{v}_j$.

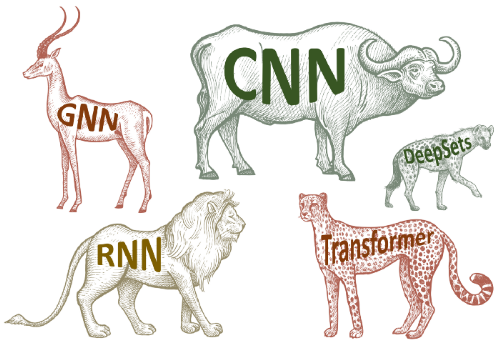


Figure 1.5: The Zoo of Neural Network Architectures

Scaled dot-product attention is not the only form of attention. There are various others such as multi-head attention which builds on scaled dot-product attention by running it in parallel multiple times. There are even task-specific attention mechanisms like those used for protein structure in AlphaFold [9].

1.1.5.5 Graph Neural Networks

Graph neural networks extend neural network models to data that is effectively represented using a graph [27]. Examples of such data include molecules, proteins, and social networks. These models are a major component of geometric deep learning with the ability to model other neural network architectures and we will discuss them in detail later. For now we only mention them and note that the power behind graph neural networks is a result of using the graph structure and updating each node's state according to its neighbors. Specifically, if we have a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and use $v_i \sim v_j$ to denote neighbors (i.e. $(i,j) \in \mathcal{E}$) and if we let $\boldsymbol{h}_i^t$ denote the hidden state of v_i at step t then we have an update of the following form (where $\boldsymbol{W}_{ij}^t$ are learnable weights):

$$\boldsymbol{h}_i^{t+1} = f\left(\sum_{j:v_i \sim v_j} c_{ij} \boldsymbol{W}_{ij}^t \boldsymbol{h}_j^t\right).$$

1.2 Motivation for Geometric Deep Learning

In the previous section, we saw a variety of different neural network architectures including feedforward neural networks, recurrent neural networks (RNNs) for sequential data, convolutional neural networks (CNNs) for grid-like data, transformers and attention, and graph neural networks (GNNs). We presented all of these models in a relatively unique manner without a unifying framework. We thus have a sort-of zoo of different neural network architectures akin to Figure 1.5 ("Geometric foundations of Deep Learning" - Michael Bronstein on Medium). A framework that unites these architectures would thus be very valuable to the field. Geometric deep learning (GDL) proposes such a framework by looking at architecture symmetries and invariants, akin to Felix Klein's framework for geometry in his *Erlangen program* [2]. We will develop and explore this framework in the remainder of this thesis and use it to analyze popular neural network architectures.

After having seen the Universal Approximation Theorem (Theorem 2) it might be unclear why we would need any neural networks other than fully connected ReLU networks. While we can approximate any "reasonable" function with a neural network the *how* is the tricky part. The number of samples required to train the neural network to approximate said function within some error is the key metric. Let us consider the class of d-dimensional 1-Lipschitz functions. The number of samples necessary to estimate this class of functions to error ε with neural networks is $O\left(\varepsilon^{-d}\right)$, which is too large in d. This is the so-called *curse of dimensionality*, showcased in Figure 1.6 (Page 9, Figure 2 of Bronstein et al. [1]).

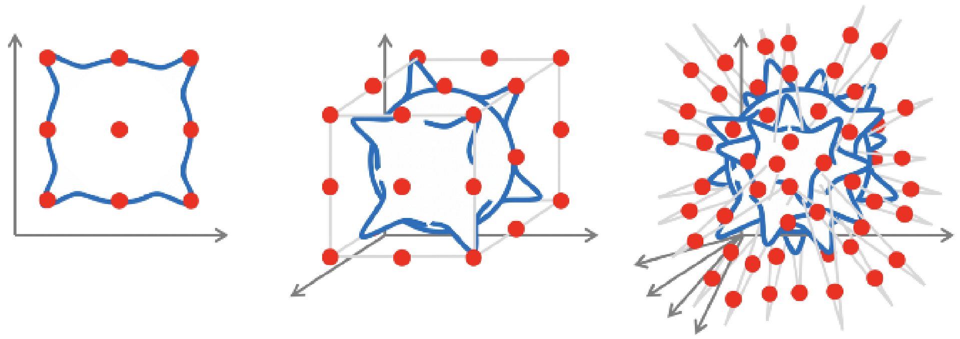


Figure 1.6: Example of the Curse of Dimensionality: Lipschitz Functions Composed of Gaussian Kernels Placed in the Quadrants of d-Dimensional Hypercubes (Blue), Samples Needed to Estimate Functions (Red)

One benefit of different neural network architectures is that they can potentially deal with the high-dimensionality of inputs. CNNs for example take high-dimensional inputs (such as images) and reduce their dimension using convolution with a kernel. This convolution creates lower-dimensional features that are easier to model by aggregating local features in the original input. Geometric deep learning models can offer a similar kind of solution to the curse of dimensionality. Symmetries and invariances in the data lower the dimension of the problem. For example, utilizing the fact that proteins are SE(3)-equivariant (i.e. they act the same if rotated and/or translated) was a key component of AlphaFold [9].

1.3 This Thesis

In the remainder of this thesis we will rigorously develop geometric deep learning, use it to view deep learning in a more unified manner, and study numerous applications. In Chapter 2, we present the basics of this framework along with relevant background in geometry. In Chapter 3, we build out the framework by looking at different geometric domains. In Chapter 4, we analyze neural network architectures. And in Chapter 5, we explore applications.

Chapter 2

The Geometric Deep Learning Blueprint

As we have seen, tackling the challenge of high-dimensional learning is a fundamental problem in machine learning. We seek to tackle this challenge using principles we refer to as *geometric priors* [1]. Fundamentally, geometric priors on our data consist of *symmetries* (invariances of our data to certain transformations) and *scale separation* (the ability to recover information from a coarser grading of our data). Many deep learning architectures utilize these principles implicitly and making them explicit will allow us to better unify the field. For now, we leave the reader with the example of CNNs, which use convolutions (translational symmetry) and pooling operations (scale separation). We first present symmetry through the lens of mathematics and in doing so provide some relevant mathematical background and terminology. We only present the most essential background content and relegate the rest of the supplementary content to the Appendix. We will proceed to use this language to explain geometric priors and to create a blueprint of geometric deep learning. The familiar reader is once again welcome and encouraged to skim and skip as they deem appropriate.

2.1 Symmetry: A Mathematical Perspective

We can think of our neural networks as operations on functions (also referred to as signals0 from a domain Ω to a space F so we naturally define the following space of functions:

Definition 4 (F-valued Functions on Ω)**.** The space of F-valued functions on Ω is

$$\mathcal{X}(\Omega, F) = \{x : \Omega \to F\}.$$

We note the vector space structure and define addition and scalar multiplication as:

$$(\alpha x + \beta y)(\omega) = \alpha x(\omega) + \beta y(\omega), \quad \alpha, \beta \in F,\ x, y \in \mathcal{X}(\Omega, F),\ \text{and } \omega \in \Omega.$$

Furthermore, if we define an inner product $\langle \cdot, \cdot \rangle_F$ on F as well as a measure μ on Ω we can define an inner product on $\mathcal{X}(\Omega, F)$ using an integral over Ω

$$\langle x, y\rangle = \int_\Omega \langle x(\omega), y(\omega)\rangle_F \, d\mu(\omega).$$

If Ω is discrete the measure μ must be discrete (likely the counting measure) and the integral becomes a sum. Note that we occasionally do not include F and write $\mathcal{X}(\Omega)$ when F is clear from the context or when F can be arbitrary.

For a practical example of this definition, consider an RGB image with resolution $n \times m$. The space of such images is $\mathcal{X}(\Omega, F)$ where $\Omega = \mathbb{Z}_n \times \mathbb{Z}_m$ ($\mathbb{Z}_k = \{0, 1, ..., k-1\}$ is the group of integers modulo k under addition) represents the grid-like structure of the image and $F = [0, 1]^3$ represents the RGB values of a pixel.

Symmetries are of great importance to us and as such we need a way to abstract the idea of a symmetry. Intuitively, two symmetries composed with each other should also be a symmetry, a symmetry should be reversible, and there should be a symmetry that leaves the system in place. These properties when made rigorous describe a *group*:

Definition 5 (Group)**.** Let $\mathfrak{G}$ be a set and $\cdot$ be a binary operator on $\mathfrak{G}$ referred to as *composition* that maps $\mathfrak{G} \times \mathfrak{G} \to \mathfrak{G}$. For $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$ we write $\mathfrak{g} \cdot \mathfrak{h} = \mathfrak{g}\mathfrak{h} \in \mathfrak{G}$ (this containment property is referred to as *closure*). We say $(\mathfrak{G}, \cdot)$ forms a group if the following properties are satisfied:

1. There exists $\mathfrak{e} \in \mathfrak{G}$ such that $\mathfrak{g}\mathfrak{e} = \mathfrak{e}\mathfrak{g} = \mathfrak{g}$ for all $\mathfrak{g} \in \mathfrak{G}$. (existence of *identity* element)
2. For every $\mathfrak{g}$ there exists $\mathfrak{h}$ such that $\mathfrak{g}\mathfrak{h} = \mathfrak{h}\mathfrak{g} = \mathfrak{e}$. We denote $\mathfrak{h} = \mathfrak{g}^{-1}$. (existence of *inverse* element)
3. For all $\mathfrak{g}, \mathfrak{h}, \mathfrak{l} \in \mathfrak{G}$ we have $(\mathfrak{g}\mathfrak{h})\mathfrak{l} = \mathfrak{g}(\mathfrak{h}\mathfrak{l})$. (associativity)

We in general write $\mathfrak{G}$ and omit the operator when it is clear from context. See Figure 2.1 ("Root of unity" - Wikipedia) for a visual example of a group.

Note that we do not in general have *commutativity*: $\mathfrak{g}\mathfrak{h} = \mathfrak{h}\mathfrak{g}$ for $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$ (it could be the case that $\mathfrak{g}\mathfrak{h} \neq \mathfrak{h}\mathfrak{g}$). For an intuitive example, note that rotating a 3-D object 90° about the xy-plane and then rotating 90° about the yz-plane is not the same as performing these rotations in the opposite order. Commutative groups are also referred to as *Abelian* groups.

Definition 6 (Subgroup)**.** Consider a group $(\mathfrak{G}, \cdot)$ and $\mathfrak{H} \subset \mathfrak{G}$. If $(\mathfrak{H}, \cdot)$ forms a group then we say it is a subgroup of $(\mathfrak{G}, \cdot)$.

Definition 7 (Group Generators)**.** If $G \subset \mathfrak{G}$ is such that every element $\mathfrak{g} \subset \mathfrak{G}$ can be written as a finite composition of elements of G and their inverses:

$$\mathfrak{g} = g_1 g_2 \cdots g_n, \; g_i \in G \text{ or } g_i^{-1} \in G$$

then we say that G generates $\mathfrak{G}$.

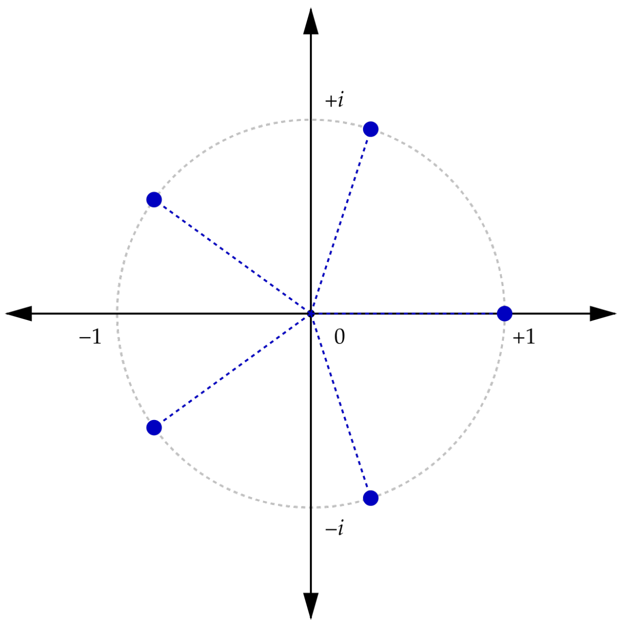


Figure 2.1: 5th Roots of Unity: We can think of the roots of unity as a group. In this case $\mathfrak{G} = \{1, \omega, \omega^2, \omega^3, \omega^4\}$ where ω is a 5th root of unity not equal to 1. The product of two roots of unity is $\omega^i \omega^j = \omega^{i+j \text{ mod } 5}$, which is analogous to addition modulo 5. We can thus think of $\mathfrak{G}$ as $\mathbb{Z}_5$.

Example 8 (Groups, Subgroups, and Generators)**.** Let $\mathfrak{G} = \mathbb{Z}_4 = \{0, 1, 2, 3\}$ be the integers modulo 4 with addition as the composition operation. Then $\mathfrak{H} = \{0, 2\}$ under the addition operation is a subgroup (in fact $\mathfrak{H}$ can be thought of as $\mathbb{Z}_2$). The group $\mathfrak{G}$ is generated by the element 1 since $1 + 1 = 2$, $2 + 1 = 3$, and $3 + 1 = 0$. Likewise, the element 2 generates the subgroup $\mathfrak{H}$ since $2 + 2 = 0$.

We note that while groups are interesting mathematical objects in their own right we are concerned with how they transform or act on our data. The following definition enables this:

Definition 9 (Group Action)**.** Given a group $\mathfrak{G}$ and a set Ω a *group action* is a mapping which assigns each group element $\mathfrak{g} \in \mathfrak{G}$ and set element $\omega \in \Omega$ to another set element. This mapping is compatible with the group operations.

$$(\mathfrak{g}, \omega) \mapsto \mathfrak{g} \cdot \omega \in \Omega \text{ such that } (\mathfrak{g}\mathfrak{h}, \omega) = \mathfrak{g}\mathfrak{h} \cdot \omega = \mathfrak{g} \cdot (\mathfrak{h} \cdot \omega) = (\mathfrak{g}, (\mathfrak{h}, \omega)).$$

For brevity, we write $\mathfrak{g} \cdot \omega = \mathfrak{g}\omega$. This definition extends to functions on Ω. Given $x \in \mathcal{X}(\Omega)$ we have

$$(\mathfrak{g} \cdot x)(\omega) = x(\mathfrak{g}^{-1}\omega).$$

We note that $(\mathfrak{g} \cdot (\mathfrak{h} \cdot x))(\omega) = ((\mathfrak{g}\mathfrak{h}) \cdot x)(\omega)$ (and recommend verifying this claim as an exercise for the inexperienced reader).

Example 10 (SE(3) Group Action)**.** $\mathfrak{G} = \mathrm{SE}(3)$ is the special Euclidean group on 3 dimensions. It consists of 3-D rotations and translations. If we let $\Omega = \mathbb{R}^3$ then the group action of $\mathfrak{G}$ on Ω is to rotate and translate the vector $\omega \in \mathbb{R}^3$.

Note that the group action on functions is linear:

$$\mathfrak{g} \cdot (\alpha x + \beta y) = \alpha(\mathfrak{g} \cdot x) + \beta(\mathfrak{g} \cdot y), \quad x, y \in \mathcal{X}(\Omega), \ \alpha, \beta \text{ scalars.}$$

We can think of a group action using a group representation:

Definition 11 (Group Representation)**.** A *real n-dimensional representation* of a group $\mathfrak{G}$ is a map $\rho : \mathfrak{G} \to \mathbb{R}^{n\times n}$ satisfying the following properties:

1. $\rho(\mathfrak{g})$ is invertible for $\mathfrak{g} \in \mathfrak{G}$
2. $\rho(\mathfrak{g}\mathfrak{h}) = \rho(\mathfrak{g})\rho(\mathfrak{h})$ for $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$

If $\rho(\mathfrak{g})$ is unitary or orthogonal we call ρ a unitary or orthogonal representation, respectively. We note that the action of $\mathfrak{G}$ on $\mathcal{X}(\Omega)$ is given by $\rho(\mathfrak{g})x(\omega) = x(\mathfrak{g}^{-1}\omega)$. Additionally, if we have $\rho : \mathfrak{G} \to \mathbb{C}^{n\times n}$ we say ρ is a *complex n-dimensional representation* of $\mathfrak{G}$.

The motivation behind our study of groups is to abstractly model symmetry. To that end, we will define functions that are either group invariant or group equivariant.

Definition 12 (Group Invariant Function)**.** We say a function $f : \mathcal{X}(\Omega) \to \mathcal{Y}$ is $\mathfrak{G}$-invariant if the group action of $\mathfrak{G}$ on the input of f does not affect its output:

$$f(\rho(\mathfrak{g})x) = f(x), \quad x \in \mathcal{X}(\Omega), \ \mathfrak{g} \in \mathfrak{G}.$$

Definition 13 (Group Equivariant Function)**.** We say a function $f : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega')$ is $\mathfrak{G}$-equivariant if the group action of $\mathfrak{G}$ on the input of f has the same affect on its output.

$$f(\rho(\mathfrak{g})x) = \rho'(\mathfrak{g})f(x), \quad x \in \mathcal{X}(\Omega), \ \mathfrak{g} \in \mathfrak{G},$$

where we note that there are distinct representations ρ and ρ' of $\mathfrak{G}$ corresponding to $\mathcal{X}(\Omega)$ and $\mathcal{X}(\Omega')$, respectively.

We have already seen an example of invariance and equivariance with convolutional neural networks (CNNs). When modeling images, shift-invariance (invariance to translations of the image) is a desired property. The convolutional layers of a CNN are actually shift-equivariant while the pooling operations are shift-invariant. We will see in general that an equivariant layer followed by an invariant layer is a common method in deep learning architectures.

2.2 Domain Structure and Stability

We are almost ready to present the blueprint but before doing so we will aim to better understand our space of problems by studying the domain.

2.2.1 More Granular Structure

We have thus far considered our domain Ω without paying attention to its structure. The amount of structure we give to Ω will depend on what our objective is. In general, coarser structures are easier to deal with computationally and we will prefer them for that reason, but for certain problems we may need to use a more granular structure. For a physical example, note that we may estimate planetary motions using Newtonian mechanics (coarse) while estimating errors in quantum computers requires atomic physics (granular).

At the coarsest level we know that Ω is a set with some elements. When we think of sets at this level we are concerned with *bijections* (Definition 21), which are invertible maps between sets (a bijection between two sets implies that they have the same number of elements).

At a more granular level, Ω could be a *topological space* (Definition 25). In this case bijections do not capture the relevant topological structure and we instead care about *homeomorphisms* (Definition 30) which are bijections between spaces that are *continuous* (Definition 29) and have continuous inverses. Intuitively, f is continuous if it maps an *open neighborhood* (Definition 26) about a point $x \in X$ to an open neighborhood about $f(x) \in f(X)$.

There is still more granularity that we can add. For example, we can add differential structure if Ω is a *differential manifold* (Definition 35), in which case we care about *diffeomorphisms* (Definition 38). These are analogous to homeomorphisms but with the requirement of being continuously differentiable (the same requirement holds for the inverse). We may also be able to consider *distance* (Definition 23) or *orientation* on our space. To think about orientation intuitively, look at your left and right hands and note that they are mirror images of each other. You cannot rotate or translate one to look like the other, you must reflect your hand to do so. And building off the idea of distance (also known as metrics), we may be able to endow a smooth manifold with a specific metric structure, making it a *Riemannian manifold* (Definition 42).

Thus far our symmetries have been thought of as maps from the domain Ω to itself. These maps are known as *automorphisms*. More generally, we are interested in *isomorphisms* (Definition 20) which are mappings between two distinct domains Ω and Ω' that show they are equivalent (an automorphism on Ω is an isomorphism between Ω and itself). An example of an automorphism would be the identity map on $X = \{0, 1\}$ (the identity map is always an automorphism and is referred to as the trivial automorphism), whereas an isomorphism would be a map between X and $Y = \{a, b\}$ that takes $0 \mapsto a$ and $1 \mapsto b$. These maps are bijections that preserve structure on sets so they can be referred to as a *set automorphism* and a *set isomorphism*, respectively.

For a less contrived example, consider the group of integers modulo 5: $\mathbb{Z}_5$ and let $\mathfrak{G}$ be the 5th roots of unity (Figure 2.1) with group operation given by multiplication. We note that for $\omega^i, \omega^j \in \mathfrak{G}$ we have $\omega^i \omega^j = \omega^{i+j \text{ mod } 5}$ and for $i, j \in \mathbb{Z}_5$ we have $i + j = (i + j) \text{ mod } 5$.

This implies that the map f which takes $i \mapsto \omega^i$ is an isomorphism between $\mathfrak{G}$ and $\mathbb{Z}_5$. In fact, f is a *group isomorphism* (Definition 22) since it respects the group structure:

$$f(i+j) = \omega^{i+j} = \omega^i \omega^j = f(i) \cdot f(j).$$

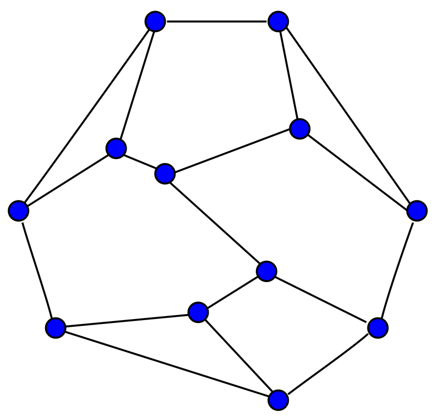
We can extend these definitions to graphs as well:

Definition 14 (Graph Isomorphism and Automorphism)**.** Given two graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ a graph isomorphism between $\mathcal{G}$ and $\mathcal{G}'$ is a map $f : \mathcal{V} \to \mathcal{V}'$ that respects the graph structure:

$$(f(u), f(v)) \in \mathcal{E}' \iff (u, v) \in \mathcal{E}.$$

A graph automorphism on G is a graph isomorphism between G and itself.

The existence of non-trivial (non-identity) automorphisms on a graph $\mathcal{G}$ implies symmetries which may be exploitable. See Figure 2.2a ("Asymmetric graph" - Wikipedia) for a graph with no automorphisms and Figure 2.2b ("Petersen graph" - Wikipedia) for a graph with multiple automorphisms.


(a) Frucht Graph

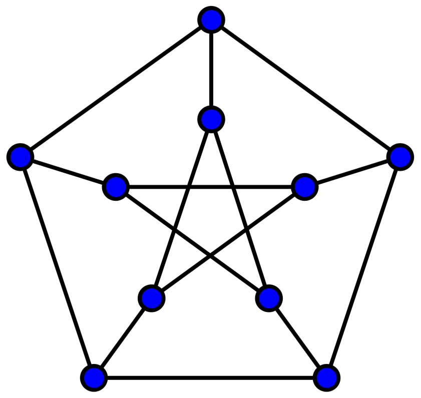

(b) Petersen Graph

Figure 2.2: (a) The Frucht graph has no automorphisms. (b) The Petersen graph has 120 automorphisms.

2.2.2 Stability and Local Symmetries

Our perspective on symmetry so far has been global in that we have considered transformations of the entire domain Ω. This is not sufficient to model the problem landscape though. Consider a scene from a movie. After a short period of time it is likely that many objects have moved from their initial frame despite the relative context of the scene staying intact. In this case, our domain Ω is fixed but we are deforming $\mathcal{X}(\Omega)$ over time. Now suppose the film switches camera angles to better highlight a specific character. This is a deformation of the domain but we still expect similar context to the previous scene. We need to properly model these local symmetries, which are more noisy and inexact compared to the global symmetries.

2.2.2.1 Function Stability

Consider an element of the hypothesis class $f \in \mathcal{F}(\mathcal{X}(\Omega))$ (the hypothesis class is the set of functions under consideration by our model). Intuitively, a small deformation of $x \in \mathcal{X}(\Omega)$ should not substantially alter our hypotheses $f(x)$. We could then consider our small deformations to be diffeomorphisms $g \in \text{Diff}(\Omega)$ that only slightly change x.

It is important to note though that composing multiple small deformations can lead to a large deformation which does alter the set of hypotheses. This motivates us to consider a specific symmetry subgroup the domain possesses: $\mathfrak{G} \subset \text{Diff}(\Omega)$ and to measure the proximity of any diffeomorphism g from this subgroup (this subgroup could correspond to translations or rotations, for example). We can measure this proximity using a complexity measure $c(g)$ which is 0 for $g \in \mathfrak{G}$. This gives us a condition for *deformation stability*:

$$\|f(\rho(g)x) - f(x)\| \leq C(c(g)\|x\|),\ \forall x \in \mathcal{X}(\Omega),$$

where C is a constant independent of x and we have the group action $\rho(g)x(\omega) = x(g^{-1}\omega)$. We call such a function x *geometrically stable*.

Since c is 0 on $\mathfrak{G}$ this definition of stability generalizes our rigid notions of group invariance and equivariance with a more flexible one. Correspondingly, the complexity is relevant to the problem and must be appropriately defined. One example for images is

$$c(g) = \left(\int_{\Omega} \|\nabla g(\omega)\|^2 \, d\omega \right)^{\frac{1}{2}},$$

which measures the proximity of g from a constant displacement (translation).

2.2.2.2 Domain Stability

Now we consider what happens when our domain Ω is deformed. Two contexts where this is common are graphs and manifolds. A graph modeling human interactions could gain or lose participants (vertices) or interactions (edges) and a manifold modeling a 3-dimensional object such as a molecule can be transformed to match a change in the molecule.

To characterize this deformation we need an appropriate metric d on the space of domains $\mathcal{D}$ (i.e. the space of graphs or of manifolds). We should have $d(\Omega, \Omega') = 0$ for equivalent Ω and Ω'. In the context of graphs, there is the graph edit distance which measures the minimum cost of vertex and edge substitutions, insertions, and deletions necessary to transform between graphs. With zero-cost substitutions this metric is zero for isomorphic graphs.

One way to construct such a distance d is using an "alignment mapping" $T : \Omega \to \Omega'$ which attempts to map between the two domains in a way that best preserves structures. For an example, let us view Riemannian manifolds (Definition 42) as metric spaces with geodesic distances (Definition 43) d_Ω and $d_{\Omega'}$ for Ω and Ω'. Let $\mathfrak{G}$ be the group of isomorphisms on $\mathcal{D}$. Then we can consider deformed functions $x \in \mathcal{X}(\Omega) \implies x' = x \circ T^{-1} \in \mathcal{X}(\Omega')$ and we have a metric:

$$d_{\mathcal{D}}(\Omega, \Omega') = \inf_{T \in \mathfrak{G}} \|d_\Omega - d_{\Omega'} \circ (T \times T)\|,$$

where we have $(d_{\Omega'} \circ (T \times T))(\omega_1, \omega_2) = d_{\Omega'}(T(\omega_1), T(\omega_2))$ for $\omega_1, \omega_2 \in \Omega$. Note that the norm (Definition 17) above is over the domain product space $\Omega \times \Omega$ and so we have converted a distance between elements of the domains Ω and Ω' to a distance on the entire domains.

With this infrastructure in place we can define *domain deformation stability*. We consider the space of functions defined over varying domains: $\mathcal{X}(\mathcal{D}) = \{(\mathcal{X}(\Omega), \Omega) : \Omega \in \mathcal{D}\}$ and say that $f : \mathcal{X}(\mathcal{D}) \to \mathcal{Y}$ is stable to domain deformations if

$$\|f(x, \Omega) - f(x', \Omega')\| \leq C\|x\| d_{\mathcal{D}}(\Omega, \Omega'), \ \forall \Omega, \Omega' \in \mathcal{D}, \ \forall x \in \mathcal{X}(\Omega).$$

2.2.2.3 Coarsening via Scale Separation

Our notion of deformation stability has allowed us to present local symmetries which strengthen our previous priors on global symmetry. However, we have still not achieved a sufficient degree of symmetry to tackle the curse of dimensionality. We have previously discussed the value of coarsening a representation and thus exploiting multiscale structure (information at different structural levels of an object). We will describe this in greater detail after a brief foray into Fourier analysis.

We can use the 1-dimensional Fourier transform to express $x(\omega) \in L^2(\Omega)$ (Definition 19) on $\Omega = \mathbb{R}$ using a linear combination of orthogonal basis functions $\varphi_\xi(\omega) = e^{i\xi\omega}$ and their frequencies ξ:

$$\hat{x}(\xi) = \int_{\mathbb{R}} x(\omega) e^{-i\xi\omega} \, d\omega.$$

We note the connection between convolution with a filter θ and the Fourier transform via the *convolution theorem*:

$$(x * \theta)(\omega) = \int_{\mathbb{R}} x(\nu)\theta(\omega - \nu) \, d\nu \implies \widehat{(x * \theta)}(\xi) = \hat{x}(\xi) \cdot \hat{\theta}(\xi).$$

The Fourier transform is able to reveal global properties of the function such as smoothness, which is useful for symmetries.

For our multiscale representation we want to decompose x into elementary functions that are local in frequency (like the Fourier transform) but also in space. We do this using a mother wavelet ψ which produces a wavelet atom via a continuous wavelet transform:

$$(W_\psi x)(\omega, \xi) = \xi^{-\frac{1}{2}} \int_{\mathbb{R}} \psi\left(\frac{\nu - \omega}{\xi}\right) x(\nu)\, d\nu.$$

Typically we sample the scale j and let $\xi = 2^{-j}$ and $\omega = k2^{-j}$ for some constant k.

The wavelet decompositions can be used as stable representations of deformations, in particular those that are high-frequency (for which Fourier decompositions are unstable). Consider a shift automorphism on Ω given by $g(\omega) = \omega - \nu$. The linear representation of the shift is $\rho(g) = S_\nu$ which shifts the phase of the Fourier transform:

$$\widehat{(S_\nu x)}(\xi) = e^{-i\xi\nu}\hat{x}(\xi).$$

We note that the Fourier modulus $f(x) = |\hat{x}|$ is shift-invariant $f(S_\nu x) = f(x)$. If we have a small translation $g(\omega) = \omega - g'(\omega)$ with $\sup_{\omega\in\Omega} \|\nabla g'(\omega)\| \leq \varepsilon$ then it can be shown that

$$\frac{\|f(\rho(g)x) - f(x)\|}{\|x\|} = O(1)$$

for arbitrarily small ε, suggesting the instability of the Fourier representation under deformations.

Wavelets on the other hand offer the necessary stability [28]:

$$\frac{\|\rho(g)(W_\psi x) - W_\psi(\rho(g)x)\|}{\|x\|} = O(\varepsilon),$$

where $\rho(g)$ acts on the spatial coordinate ω of $(W_\psi x)(\omega, \xi)$. We note that we have an approximate equivariance above and not yet an invariance.

We will aim to coarsen our domain Ω at different scales into a hierarchy $\Omega_1, \Omega_2, ..., \Omega_J$. At a high-level, a coarsening groups nearby points in Ω, which implies the necessity of a metric over the domain. We can extract a notion of *local stability* from this coarsening. Letting $\mathcal{X}_j(\Omega_j, F_j) = \{x_j : \Omega_j \to F_j\}$ we call $f : \mathcal{X}(\Omega) \to \mathcal{Y}$ locally stable at scale j if there exists a function $f_j : \mathcal{X}_j(\Omega_j) \to \mathcal{Y}$ at scale j and a non-linear coarsening $C_j : \mathcal{X}(\Omega) \to \mathcal{X}_j(\Omega_j)$ to scale j such that $f \approx f_j \circ C_j$. This definition essentially says that the long-range interactions that f depends on can be determined from localised interactions at scale j.

Coarsening via scale separation is fundamental in machine learning and shows up as local pooling in convolutional neural networks and graph neural networks, as we will see.

2.3 The Blueprint

We now combine the ideas developed so far into a general framework for high-dimensional learning by looking at functions in $\mathcal{X}(\Omega, F)$ with domain Ω that has symmetry group $\mathfrak{G}$.

We have not yet arrived at a single architecture but our observations suggest a construction. The first thing to note is that for expressivity we require non-linear functions f since otherwise by $\mathfrak{G}$-invariance we have:

$$f(x) = \frac{1}{\mu(\mathfrak{G})}\int_{\mathfrak{G}} f(x)\, d\mu(\mathfrak{g}) = \frac{1}{\mu(\mathfrak{G})}\int_{\mathfrak{G}} f(\mathfrak{g}x)\, d\mu(\mathfrak{g}) = f\left(\frac{1}{\mu(\mathfrak{G})}\int_{\mathfrak{G}} (\mathfrak{g}x)\, d\mu(\mathfrak{g})\right),\ \forall x \in \mathcal{X}(\Omega),$$

where $\mu(\mathfrak{g})$ is the *Haar measure* (Definition 44) of the group $\mathfrak{G}$ (which we can think of as assigning a volume to subsets of $\mathfrak{G}$) and $\mu(\mathfrak{G}) = \int_{\mathfrak{G}} d\mu(\mathfrak{g})$. The equality above indicates that f depends on the group average of x over $\mathfrak{G}$, which is a disastrous result (imagine if we represented every pixel of an image by the average pixel).

This suggests that linear invariant functions are not very powerful objects. However, linear equivariant functions do not have such constraints and are much more useful especially since we can compose them with certain non-linear maps. Specifically, take a $\mathfrak{G}$-equivariant map $f : \mathcal{X}(\Omega, F) \to \mathcal{X}(\Omega, F')$ and an arbitrary (potentially non-linear) map $\sigma : F' \to F''$. The composition $\ell = \boldsymbol{\sigma} \circ f : \mathcal{X}(\Omega, F) \to \mathcal{X}(\Omega, F'')$, where $\boldsymbol{\sigma} : \mathcal{X}(\Omega, F') \to \mathcal{X}(\Omega, F'')$ indicates σ applied element-wise (i.e. $(\boldsymbol{\sigma}(f))(\omega) = \sigma(f(\omega))$), is $\mathfrak{G}$-equivariant as well.

We can use this property to construct $\mathfrak{G}$-invariant functions that take the general form $M_{\mathfrak{G}} \circ g : \mathcal{X}(\Omega, F) \to F''$, where $M_{\mathfrak{G}}$ indicates taking an average over $\mathfrak{G}$. Adapting universal approximation theorem like arguments shows that we can approximate any $\mathfrak{G}$-invariant function via this construction. We are motivated to consider localised equivariant maps based on our comparison of Fourier and wavelet representations (which showed a tradeoff between global invariance and deformation stability). Given a distance on Ω we say that f is localised if $(f(x))(\omega)$ is a function only of $x(\nu)$ for ν contained in the *receptive field* $\mathcal{N}_\omega = \{\nu : d(\omega, \nu) \leq r\}$ with r a small radius. Note that we cannot approximate functions with long-range interactions using a single equivariant map g, however we can compose several of them to do so: $g_J \circ g_{J-1} \circ \cdots \circ g_2 \circ g_1$. We additionally coarsen the domain between each equivariant map. This construction allows us to increase the receptive field.

The geometric deep learning blueprint, which we will now present, consists of three components discussed and developed in this chapter: local equivariant maps, a global invariant map, and coarsening operators.

Definition 15 (The Geometric Deep Learning Blueprint)**.** Take two domains Ω and Ω' and $\mathfrak{G}$ a symmetry group over Ω. Consider the following components:

1. Linear $\mathfrak{G}$-equivariant layers: $f : \mathcal{X}(\Omega, F) \to \mathcal{X}(\Omega', F')$ with $f(\mathfrak{g}x) = \mathfrak{g}f(x)$ for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega, F)$

2. Non-linear activation functions: $\sigma : F \to F'$ applied element-wise

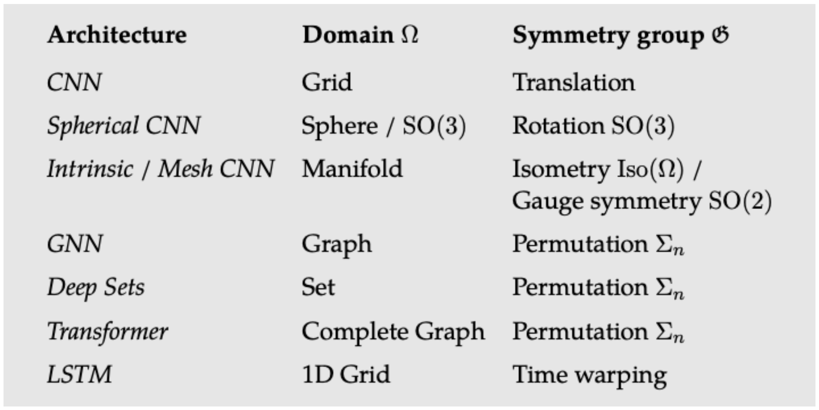
3. Coarsening (local pooling) operators: $P : \mathcal{X}(\Omega, F) \to \mathcal{X}(\Omega', F)$ where Ω' is a compact version of Ω: $\Omega' \subseteq \Omega$

4. $\mathfrak{G}$-invariant layers (global pooling): $M_{\mathfrak{G}} : \mathcal{X}(\Omega, F) \to \mathcal{Y}$ with $M_{\mathfrak{G}}(\mathfrak{g}x) = M_{\mathfrak{G}}(x)$ for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega, F)$

With these components we construct a $\mathfrak{G}$-invariant function $g : \mathcal{X}(\Omega, F) \to \mathcal{Y}$:

$$g = M_{\mathfrak{G}} \circ \sigma_J \circ f_J \circ P_{J-1} \circ \cdots \circ P_1 \circ \sigma_1 \circ f_1,$$

where each component has appropriate input and output spaces. Note that different components may exploit different symmetry groups $\mathfrak{G}$.



Architecture	**Domain Ω**	**Symmetry group $\mathfrak{G}$**
CNN	Grid	Translation
Spherical CNN	Sphere / SO(3)	Rotation SO(3)
Intrinsic / Mesh CNN	Manifold	Isometry Iso(Ω) / Gauge symmetry SO(2)
GNN	Graph	Permutation Σ_n
Deep Sets	Set	Permutation Σ_n
Transformer	Complete Graph	Permutation Σ_n
LSTM	1D Grid	Time warping

Figure 2.3: Blueprint Overview for Popular Neural Network Architectures

The above blueprint enables significant generality. In Chapter 3, we will investigate the different geometric domains and in Chapter 4, we will apply the blueprint to different neural network architectures. For now, we provide a preview by highlighting several architectures, their domains, and their symmetry groups in Figure 2.3 (Page 30 of Bronstein et al. [1]).

Chapter 3

Geometric Domains: 5 Gs and an M

We now focus our discussion on the different geometric domains that we can apply geometric deep learning to. We focus on 5 Gs: graphs, grids, groups (thought of as symmetries in a homogeneous space), geodesics on manifolds, and gauges on tangent bundles. We also focus on an M: meshes. We will define these terms throughout the chapter, especially for the latter two Gs which are far more abstract than the others.

3.1 Graphs

Graphs are useful objects in many modeling contexts ranging from human behavior to molecules. Furthermore, they are permutation-invariant, which allows us to consider symmetries on sets which, in addition to grids, are a special case of graphs. For sake of completion we give a formal definition of graphs:

Definition 16 (Graph)**.** A graph is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ a set of *vertices* (or *nodes*) and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ a set of *edges* between vertices. We additionally endow each vertex $v \in \mathcal{V}$ with vertex features $\boldsymbol{x}_v \in \mathbb{R}^d$.

We can think of a social network as a graph where the vertices are the users, the edges are friendships or followings, and the vertex features are user characteristics (name, age, location, interests, ...). We can also think of a molecule as a graph where the vertices are the atoms, the edges are bonds, and the vertex features are atomic properties.

Note that the vertices of a graph are unordered which implies that graphs are permutation-invariant at the vertex level (i.e. a graph $\mathcal{G}$ and a graph $\mathcal{G}'$ derived by permuting the vertices of $\mathcal{G}$ are isomorphic) and so functions defined on a graph should also be permutation-invariant. Considering the context of the blueprint(Definition 15) we have domain $\Omega = \mathcal{G}$, function space $\mathcal{X}(\mathcal{G}, \mathbb{R}^d)$, and symmetry group $\mathfrak{G} = \Sigma_n$ where Σ_n is the *permutation group* on n elements which consists of all orderings of the set $\{1, 2, ..., n\}$ (we have assumed $|\mathcal{V}| = n$ and that i indexes $v_i \in \mathcal{V}$).

To demonstrate *permutation invariance* we consider sets. We can think of the set $\mathcal{V}$ as the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{E} = \emptyset$. We can order the vertices of our graph by creating a matrix using their features:

$$\boldsymbol{X} = \begin{pmatrix} — & \boldsymbol{x}_1 & — \\ — & \boldsymbol{x}_2 & — \\ & \vdots & \\ — & \boldsymbol{x}_n & — \end{pmatrix} \in \mathbb{R}^{n \times d}.$$

The action of a permutation $\mathfrak{g} \in \Sigma_n$ acting on $\boldsymbol{X}$ permutes the rows of the matrix which can be represented using a *permutation matrix* $\boldsymbol{P} = \rho(\mathfrak{g})$, which is a matrix of all zeroes except for a single 1 in each row and each column. A function f with domain $\mathbb{R}^{n \times d}$ is permutation-invariant if $f(\boldsymbol{PX}) = f(\boldsymbol{X})$. A general example of such an f is:

$$f(\boldsymbol{X}) = \phi\left(\sum_{v \in \mathcal{V}} \psi(\boldsymbol{x}_v)\right).$$

We note that ψ is applied to each vertex's features independently and ϕ is applied to a sum over all vertices. The sum is permutation-invariant and so f is permutation-invariant.

This invariance again has the limitation of only providing global graph information, whereas we may be interested in local information. If a (vertex-level) function $\boldsymbol{F}(\boldsymbol{X})$ were to transform the vertex features in the matrix $\boldsymbol{X}$ then we will want to connect the rows of $\boldsymbol{X}$ to the rows of $\boldsymbol{F}(\boldsymbol{X})$ to understand how the vertex-level features are being modified. This leads us to a definition of *permutation equivariance* where $\boldsymbol{F}(\boldsymbol{PX}) = \boldsymbol{PF}(\boldsymbol{X})$. The linear transformation $\boldsymbol{F}_{\boldsymbol{\Theta}}(\boldsymbol{X}) = \boldsymbol{X\Theta}$ where $\boldsymbol{\Theta} \in \mathbb{R}^{d \times \ell}$ is an example of a permutation-equivariant function which transforms the features $\boldsymbol{x}_v$ into $\boldsymbol{\Theta}^{\mathsf{T}} \boldsymbol{x}_v$.

We now extend our definitions of permutation invariance and equivariance to graphs in general. Let $\boldsymbol{A} \in \{0, 1\}^{n \times n}$ be the adjacency matrix of $\mathcal{G}$ defined such that

$$\boldsymbol{A}_{uv} = \begin{cases} 1 & (u, v) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

The matrix $\boldsymbol{X}$ encodes the vertex level information of $\mathcal{G}$ while the adjacency matrix $\boldsymbol{A}$ encodes the edge level information. A function on our graph then becomes a function of $\boldsymbol{X}$ and $\boldsymbol{A}$. Note that applying a permutation to $\mathcal{G}$ results in a corresponding permutation matrix $\boldsymbol{P}$ being applied to both the rows and columns of $\boldsymbol{A}$ producing $\boldsymbol{PAP}^{\mathsf{T}}$. We then say that a (graph-level) function f is permutation-invariant if for any permutation matrix $\boldsymbol{P}$

$$f(\boldsymbol{PX}, \boldsymbol{PAP}^{\mathsf{T}}) = f(\boldsymbol{X}, \boldsymbol{A})$$

and a (vertex-level) function $\boldsymbol{F}$ is permutation-equivariant if for any permutation matrix $\boldsymbol{P}$

$$\boldsymbol{F}(\boldsymbol{PX}, \boldsymbol{PAP}^{\mathsf{T}}) = \boldsymbol{PF}(\boldsymbol{X}, \boldsymbol{A}).$$

We now attempt to generate permutation-equivariant functions using the blueprint and vertex neighborhoods which express locality. We define a neighborhood on $v \in \mathcal{V}$ as:

$$\mathcal{N}_v = \{u \in \mathcal{V} : (u, v) \in \mathcal{E} \text{ or } (v, u) \in \mathcal{E}\}.$$

Note that our neighborhood is undirected. Let the neighborhood features of v be the multiset (a set that can have duplicate elements)

$$\boldsymbol{X}_{\mathcal{N}_v} = \{\{\boldsymbol{x}_u : u \in \mathcal{N}_v\}\}.$$

Take ϕ a function of a vertex's features and its neighborhood features $\phi(\boldsymbol{x}_v, \boldsymbol{X}_{\mathcal{N}_v})$. If we construct ϕ so that it is permutation-invariant then we can use it to create a permutation-equivariant function:

$$\boldsymbol{F}(\boldsymbol{X}, \boldsymbol{A}) = \begin{pmatrix} — & \phi(\boldsymbol{x}_1, \boldsymbol{X}_{\mathcal{N}_1}) & — \\ — & \phi(\boldsymbol{x}_2, \boldsymbol{X}_{\mathcal{N}_2}) & — \\ & \vdots & \\ — & \phi(\boldsymbol{x}_n, \boldsymbol{X}_{\mathcal{N}_n}) & — \end{pmatrix}.$$

To prove permutation equivariance, note that we assumed ϕ to be permutation-invariant. Therefore, applying a permutation $\boldsymbol{P}$ does not alter the values of ϕ. However, the permutation does reorder the vertices thus outputting $\boldsymbol{PF}(\boldsymbol{X}, \boldsymbol{A})$, as desired.

It is important to note that unlike sets and grids (special cases of graphs) which only rely on vertex features, graphs in general take into consideration a variable structure on the domain and do not assume fixed relationships. Furthermore, the structure of graphs and grids, unlike sets, allow for non-trivial coarsenings.

3.2 Grids

Now we investigate grids, which have been incredibly valuable to machine learning given the successes in computer vision and natural language processing. Grids are indeed a special case of graphs but their fixed adjacency permits the stronger geometric prior of translation invariance.

We had earlier mentioned that convolution can be represented as matrix multiplication using circulant matrices. We can think of a 1-dimensional grid as a group with cyclic structure that has vertices indexed by $\mathbb{Z}_n$ (integers under addition modulo n). Each node $v \in \mathbb{Z}_n$ has a left and right neighbor, namely $v - 1$ and $v + 1$, respectively. We then have $\phi(\boldsymbol{x}_v, \boldsymbol{X}_{\mathcal{N}_v}) = \phi(\boldsymbol{x}_{v-1}, \boldsymbol{x}_v, \boldsymbol{x}_{v+1})$. If ϕ is linear then $\phi(\boldsymbol{x}_{v-1}, \boldsymbol{x}_v, \boldsymbol{x}_{v+1}) = \theta_{-1}\boldsymbol{x}_{v-1} + \theta_0\boldsymbol{x}_v\theta_1\boldsymbol{x}_{v+1}$ and we have a translation-equivariant function:

$$\boldsymbol{F}(\boldsymbol{X}) = \begin{pmatrix} \theta_0 & \theta_1 & & & \theta_{-1} \\ \theta_{-1} & \theta_0 & \theta_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \theta_{-1} & \theta_0 & \theta_1 \\ \theta_1 & & & \theta_{-1} & \theta_0 \end{pmatrix} \begin{pmatrix} — & \boldsymbol{x}_0 & — \\ — & \boldsymbol{x}_1 & — \\ & \vdots & \\ — & \boldsymbol{x}_{n-1} & — \end{pmatrix}.$$

The matrix on the left in the above expression is a *circulant matrix* $\boldsymbol{C}(\boldsymbol{\theta}) = [\theta_{i-j \bmod n}]_{0\le i,j\le n-1}$ where $\boldsymbol{\theta} = (\theta_0, \theta_1, ..., \theta_{n-1})$. Circulant matrices have a single value across each "diagonal" of the matrix, which offers weight sharing (also known as parameter sharing). The connection with discrete convolutions follows from the fact that:

$$(\boldsymbol{x} * \boldsymbol{\theta})_v = \sum_{u=0}^{n-1} x_{u \bmod n} \cdot \theta_{v-u \bmod n} = (\boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{x})_v \implies \boldsymbol{x} * \boldsymbol{\theta} = \boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{x}.$$

Note that for $\boldsymbol{\theta} = (0,1,0,0,...)^\mathsf{T}$ the circulant matrix $\boldsymbol{S} = \boldsymbol{C}(\boldsymbol{\theta})$ shifts vectors to the right by one position. We call $\boldsymbol{S}$ a *shift operator* (or *translation operator*). Note that circulant matrices are commutative: $\boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{C}(\boldsymbol{\omega}) = \boldsymbol{C}(\boldsymbol{\omega})\boldsymbol{C}(\boldsymbol{\theta})$. From this we derive the shift equivariance (or translation equivariance) of convolution:

$$\boldsymbol{S}\boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{x} = \boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{S}\boldsymbol{x}.$$

We now use these results to show that the Fourier transform diagonalizes the convolution operator. We will rely on the well-known fact from linear algebra that two matrices A, B are simultaneously diagonalizable (i.e. there exists an invertible matrix S such that $S^{-1}AS$ and $S^{-1}BS$ are both diagonal) if and only if they commute. In particular, this result implies the existence of a common eigenbasis for the circulant matrices. We then compute this eigenbasis using a single circulant matrix. The shift operator S has an eigenbasis that coincides with the discrete Fourier basis:

$$\boldsymbol{\varphi}_k = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & e^{\frac{2\pi i k}{n}} & e^{\frac{4\pi i k}{n}} & \dots & e^{\frac{2\pi i (n-1)}{n}} \end{pmatrix}^\mathsf{T}, \; k \in \{0, 1, ..., n-1\}.$$

Define the Fourier matrix $\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\varphi}_0 & \boldsymbol{\varphi}_1 & \dots & \boldsymbol{\varphi}_{n-1} \end{pmatrix}$ and note that the discrete Fourier transform (DFT) can be written using this matrix:

$$(\boldsymbol{\Phi}^*\boldsymbol{x})_k = \hat{x}_k = \frac{1}{\sqrt{n}} \sum_{v=0}^{n-1} x_v e^{-\frac{2\pi i k v}{n}} \text{ (DFT) and } (\boldsymbol{\Phi}\hat{x})_v = x_v = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \hat{x}_k e^{\frac{2\pi i k v}{n}} \text{ (Inverse DFT).}$$

Now using the fact that the eigenvalues of $\boldsymbol{C}(\boldsymbol{\theta})$ are equal to $\hat{\boldsymbol{\theta}} = \boldsymbol{\Phi}^*\boldsymbol{\theta}$ [29] we derive the convolution theorem:

$$\boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{x} = \boldsymbol{\Phi} \cdot \operatorname{diag}(\hat{\theta}_0, \hat{\theta}_1, ..., \hat{\theta}_{n-1}) \cdot \boldsymbol{\Phi}^*\boldsymbol{x} = \boldsymbol{\Phi}(\hat{\boldsymbol{\theta}} \odot \hat{\boldsymbol{x}}),$$

where $\odot$ indicates element-wise multiplication.

We end our discussion of grids with a derivation of the continuous Fourier transform, which will be useful to us in other contexts. Take f defined on $\mathbb{R}$ and consider the shift operator S_v defined such that $(S_v f)(u) = f(u-v)$. Consider also the Fourier basis functions $\varphi_\xi(u) = e^{i\xi u}$. Then we have $S_v\varphi_\xi(u) = S_v e^{i\xi u} = e^{i\xi(u-v)} = e^{-i\xi v}e^{i\xi u} = e^{-i\xi v}\varphi_\xi(u)$ implying that the basis functions are complex eigenvectors of S_v with eigenvalues $e^{-i\xi v}$. Note that

$\|S_v f\| = \|f\|$ implying that the eigenvalues of S_v have magnitude 1. Note also that any two functions with the same eigenvalue must be collinear. To see this note that if $S_v f = e^{-i\xi_0 v} f$ then taking the Fourier transform we have

$$e^{-i\xi v}\hat{f}(\xi) = e^{-i\xi_0 v}\hat{f}(\xi),\ \forall \xi \in \mathbb{R}.$$

This implies that $f(\xi) = 0$ for $\xi \neq \xi_0$ which implies that $f = ce^{-i\xi_0 v} = c\varphi_{\xi_0}$.

Now take C a linear translation-equivariant operator: $S_v C = C S_v$. Then we have

$$(S_v C)e^{i\xi u} = (CS_v)e^{i\xi u} = e^{-i\xi v} C e^{i\xi u},$$

which suggests that $Ce^{i\xi u}$ is an eigenfunction of S_v with eigenvalue $e^{-i\xi v}$, implying that the Fourier basis is the eigenbasis of all translation-equivariant operators. C is then diagonal in the Fourier domain (the system of coordinates rotated by the Fourier transform) and can be expressed as $Ce^{i\xi u} = \hat{p}_C(\xi)e^{i\xi u}$ with $\hat{p}_C$ a *transfer function* on different frequencies. Now for a function x we have

$$(Cx)(u) = C\left(\int_{\mathbb{R}} \hat{x}(\xi)e^{i\xi u}\, d\xi\right) = \int_{\mathbb{R}} \hat{x}(\xi)\hat{p}_C(\xi)e^{i\xi u}\, d\xi$$

$$= \int_{\mathbb{R}} p_C(v)x(u-v)\, dv = (x * p_C)(u).$$

This result implies that every linear translation equivariant operator is a convolution.

Two final notes: The Fourier transform can be computed in $O(n \log n)$ time using a fast Fourier transform (FFT) algorithm (which exploits the structure of Φ). The graph Fourier transform and an analogy to convolution for graphs can be defined using the eigenvectors of said graph's adjacency matrix [30]. Graph neural networks (GNNs) designed using these convolutions can be referred to as "spectral GNNs."

3.3 Groups

In our previous section on grids we proved an important result, namely an equivalence between convolutions and shift-equivariant linear operators. Additionally, we showed that shift operators are simultaneously diagonalizable via the Fourier transform. In this section, we will generalize these results to arbitrary symmetry groups that we can integrate (or sum) over. If our domain is the real line $\mathbb{R}$ then we can think of the convolution of a function x and a filter θ as matching the input signal with shifted copies of the filter:

$$(x * \theta)(u) = \langle x, S_u\theta\rangle = \int_{\mathbb{R}} x(v)\theta(u+v)\, dv.$$

Here the symmetry group is $\mathfrak{G} = \mathbb{R}$. We can generalize these ideas by looking at different groups $\mathfrak{G}$. We briefly note that the above expression is really the *cross-correlation*, which in the field of deep learning goes by convolution.

Now we define *group convolution*. Recall that the action of $\mathfrak{g} \in \mathfrak{G}$ on $x \in \mathcal{X}(\Omega)$ is given by $\rho(\mathfrak{g})x(\omega) = x(\mathfrak{g}^{-1}\omega)$ where ρ is a representation of $\mathfrak{G}$. Previously, we dealt with the translation group $\mathfrak{G} = \mathbb{R}$ where $v \in \mathbb{R}$ had action of shifting the coordinates by $+v$. Correspondingly, the representation is the shift operator $\rho(v) = S_v$ with $(S_v f)(u) = f(u-v)$. We now assume that $\mathcal{X}(\Omega)$ is a Hilbert space and consider the inner product:

$$\langle x, \theta \rangle = \int_\Omega x(\omega)\theta(\omega)\, d\omega,\ x \in \mathcal{X}(\Omega, \mathbb{R}).$$

Note that our definition holds only for scalar functions on Ω. In general, for higher-dimensional x we replace the product in the integral above with an inner product $\langle x(\omega), \theta(\omega)\rangle$ in the appropriate space. We now define for $x \in \mathcal{X}(\Omega, \mathbb{R})$ the group convolution (which takes elements of $\mathfrak{G}$ as inputs):

$$(x * \theta)(\mathfrak{g}) = \langle x, \rho(\mathfrak{g})\theta \rangle = \int_\Omega x(\omega)\theta(\mathfrak{g}^{-1}\omega)\, d\omega.$$

The group convolution is $\mathfrak{G}$-equivariant since for any $\mathfrak{h} \in \mathfrak{G}$ and $x \in \mathcal{X}(\Omega)$ we have

$$(\rho(\mathfrak{h})x * \theta)(\mathfrak{g}) = \langle \rho(\mathfrak{h})x, \rho(\mathfrak{g})\theta \rangle = \langle x, \rho(\mathfrak{h}^{-1}\mathfrak{g})\theta \rangle = \rho(\mathfrak{h})(x * \theta)(\mathfrak{g}),$$

where the first equality follows by definition, the second follows from the fact that $\langle x, \rho(\mathfrak{g})\theta \rangle = \langle \rho(\mathfrak{g}^{-1})x, \theta \rangle$ (since matching x with the $\mathfrak{g}$-transformed filter θ is equivalent to matching the inverse $\mathfrak{g}$-transformed x with θ) and because $\rho(\mathfrak{h}^{-1}\mathfrak{g}) = \rho(\mathfrak{h}^{-1})\rho(\mathfrak{g})$, and the third equality follows from the definition of the group action on θ.

For an example we have already worked with, consider the 1-dimensional grid: $\Omega = \mathbb{Z}_n$ and $\mathfrak{G} = \mathbb{Z}_n$. We note that the group action is a shift modulo n. If we identify $\mathfrak{g} \in \mathfrak{G}$ with $u \in \{0, 1, ..., n-1\}$ then $\mathfrak{g}v = v - u \bmod n$ and $\mathfrak{g}^{-1}v = v + u \bmod n$. Our group convolution is then

$$(x * \theta)(\mathfrak{g}) = \int_{\mathbb{Z}_n} x(\omega)\theta(\mathfrak{g}^{-1}\omega)\, d\omega = \sum_{v=0}^{n-1} x_v \theta_{\mathfrak{g}^{-1}v} = \sum_{v=0}^{n-1} x_v \theta_{v+u \bmod n}.$$

Note that the sum arises from the fact that the integral over the discrete group $\mathbb{Z}_n$ is with respect to the counting measure. The above expression coincides with our definition of convolution.

For a more interesting example we consider *spherical convolution*, which is relevant in fields such as chemistry and molecular biology. Our domain is the sphere in 3-dimensions:

$$\Omega = S^2 = \{x \in \mathbb{R}^3 : \|x\| = 1\}$$

and our symmetry group is the *special orthogonal group* on 3 dimensions which consists of rotations represented by orthogonal matrices with determinant 1

$$\mathfrak{G} = \mathrm{SO}(3) = \{\boldsymbol{R} \in \mathbb{R}^{3\times 3} : \boldsymbol{R}^\mathsf{T}\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^\mathsf{T} = \mathbb{1},\ \det \boldsymbol{R} = 1\}.$$

Note that the det $\boldsymbol{R} = 1$ condition implies that the transformations are orientation-preserving (the orthogonal group $\mathrm{O}(3)$ does not require this).

We see that the action of the group $\mathrm{SO}(3)$ is given by an orthogonal matrix $\boldsymbol{R}$ with determinant 1. For $x \in \mathcal{X}(S^2)$ we can define the spherical convolution as

$$(x * \theta)(\boldsymbol{R}) = \int_{S^2} x(\boldsymbol{\omega})\theta(\boldsymbol{R}^{-1}\boldsymbol{\omega})\, d\boldsymbol{\omega}.$$

The spherical convolution is a function of the group $\mathrm{SO}(3)$, which is a 3-manifold and in fact a Lie group (Definitions 31 and 36) and not a function of the domain S^2, which is a 2-manifold. This has important implications in the context of the blueprint since we now must deal with $\mathcal{X}(\mathfrak{G})$, which in our case is $\mathcal{X}(\mathrm{SO}(3))$. We can do this by considering group convolution on the domain $\Omega = \mathfrak{G}$ with symmetry group $\mathfrak{G}$ where the group action is given by the group operation. We then have $(\rho(\mathfrak{g})x)(\mathfrak{h}) = x(\mathfrak{g}^{-1}\mathfrak{h})$ for $x \in \mathcal{X}(\mathfrak{G})$. For spherical convolution we convolve with another filter ϕ to find

$$((x * \theta) * \phi)(\boldsymbol{R}) = \int_{\mathrm{SO}(3)} (x * \theta)(\boldsymbol{Q}) \cdot \phi(\boldsymbol{R}^{-1}\boldsymbol{Q})\, d\boldsymbol{Q}.$$

We note that group convolution involves integration over the domain Ω, which for the sake of tractability requires Ω to have small cardinality if discrete or low-dimension in the continuous case. We can thus do convolution over domains such as $\mathrm{SO}(3)$ or $\mathbb{R}^2$, whereas convolution over the permutation group Σ_n (which has $n!$ elements) or the affine group is not feasible. However, we may still build convolutions for large groups by considering their actions on low-dimensional domains.

We have made an implicit assumption when dealing with domains such as the Euclidean space, grids, and spheres. For any of these domains, we can transform a point in said domain to any other point in the domain. We call such a domain Ω a *homogeneous space* if for any $\omega, \omega' \in \Omega$ there exists $\mathfrak{g} \in \mathfrak{G}$ such that $\mathfrak{g}\omega = \omega'$. We seek to move away from this assumption in our next discussion.

3.4 Geodesics

In our discussion of the spherical convolution we mentioned the manifolds S^2 and $\mathrm{SO}(3)$. These manifolds have global symmetries but that is not the case for most manifolds. Manifolds have two kinds of invariances that we can exploit: metric structure preserving transformations and local reference frame changes.

The term "manifold" is thrown around in many contexts and thus may seem like a particularly complicated object. In fact, they model a number of objects in fields ranging from structural biology to 3-dimensional objects in computer vision and augmented/virtual reality. Manifolds offer compact representations that ignore internal structures and allow for domain deformations (such as the dynamics of a protein's structure).

We will seek to apply the blueprint to manifolds, but to do this we must first cover the basic mathematical theory underlying manifolds. We present this here somewhat informally and include formal definitions in the Appendix. The familiar reader is of course welcome and encouraged to skim or skip ahead to the sections on Fourier analysis and convolution on manifolds.

3.4.1 Manifold Basics

An *n-manifold* (Definition 31) is a space Ω for which there exists a *neighborhood* (Definition 26) around each point that is *homeomorphic* (Definition 30) to $\mathbb{R}^n$ (i.e. the space can be mapped to $\mathbb{R}^n$ while preserving structure). We may also call such a space an *n-dimensional manifold*. A *differential manifold* (Definition 35) also known as a *smooth manifold* is a manifold with additional differential structure allowing for differential calculus. Manifolds can be locally approximated around any point by $T_\omega\Omega$ the *tangent space* (Definition 39), which can be thought of as an n-dimensional plane attached to the point ω. The collection of tangent spaces forms $T\Omega$ the *tangent bundle* (Definition 41). An element of the tangent space $X \subset T_\omega\Omega$ is called a *tangent vector* and can be thought of as a displacement from the point ω. See Appendix Figure 5.4 ("Tangent Space" - Wikipedia) for visuals of the tangent space and its tangent vectors and see Appendix Figure 5.5 ("Tangent Bundle" - Wikipedia) for visuals of the tangent bundle of the circle S_1.

In order to measure angles between tangent vectors and their lengths we require a *Riemannian metric g*, which at each point $\omega \in \Omega$ admits a smooth positive definite bilinear function:

$$g_\omega : T_\omega\Omega \times T_\omega\Omega \to \mathbb{R}, \quad g_\omega(X, X) > 0 \text{ for all } X \in T_\omega\Omega \text{ such that } X \neq 0.$$

We can use the Riemannian metric to measure angles and lengths via the following inner product and norm, respectively:

$$\langle X, Y\rangle_\omega = g_\omega(X, Y), \quad \|X\|_\omega = \sqrt{g_\omega(X, X)}, \quad X, Y \in T_\omega\Omega.$$

Note that tangent vectors do not have coordinates and are instead abstract entities. A tangent vector X can only be expressed as a list of coordinates $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ relative to a local basis $\{X_1, X_2, ..., X_n\} \subset T_\omega\Omega$. The metric g_ω can only be expressed in this way as well with an $n \times n$ matrix $\boldsymbol{G}$ such that $g_{ij} = g_\omega(X_i, X_j)$.

A *Riemannian manifold* (Definition 42) is a smooth manifold with a Riemannian metric. Properties of the manifold that depend only on the metric are called *intrinsic properties*. These properties are of interest since they are isometry-invariant meaning that isometric deformations will not alter them. We also note, as an aside, the result of the *Nash embedding theorem* [31] which states that any smooth Riemannian manifold can be embedded into $\mathbb{R}^d$

for sufficiently large d.

We call a function $x : \Omega \to \mathbb{R}$ a *scalar field* and note that scalar fields form a vector space $\mathcal{X}(\Omega, \mathbb{R})$ with an inner product:

$$\langle x, y \rangle = \int_\Omega x(\omega) y(\omega) \, d\omega$$

with $d\omega$ the volume measure induced by the Riemannian metric. A function $X : \Omega \to T\Omega$ that maps each $\omega \in \Omega$ to $F(\omega) \in T_\omega \Omega$ is called a *tangent vector field.* These functions form a vector space $\mathcal{X}(\Omega, T\Omega)$ with inner product defined using the Riemannian metric:

$$\langle X, Y \rangle = \int_\Omega g_\omega(X(\omega), Y(\omega)) \, d\omega.$$

We want to define vector fields by generalizing derivatives. Consider the classic *differential* $dx(\omega) = x(\omega + d\omega) - x(\omega)$. We unfortunately cannot apply this definition to manifolds since they do not, in general, have a vector space structure. We can think of a vector field as a map $F : \mathcal{X}(\Omega, \mathbb{R}) \to \mathcal{X}(\Omega, \mathbb{R})$ with the following properties for any $x, y \in \mathcal{X}(\Omega, \mathbb{R})$:

1. $F(c) = 0$ for constants c
2. $F(x + y) = F(x) + F(y)$ (linearity)
3. $F(xy) = F(x)y + xF(y)$ (product rule)

Given $x \in \mathcal{X}(\Omega, \mathbb{R})$ we can consider the differential $dx(F) = F(x)$ as an extension of the *directional derivative.* To see this note that we have a map $(x, F) \to F(x)$, which can be interpreted as the displacement $F \in T_\omega\Omega$ displacing x by $dx_\omega(F)$.

We can also think of the differential as a linear function on tangent vectors $dx_\omega : T_\omega\Omega \to \mathbb{R}$ which, assuming we have a Riemannian metric, we can express as

$$dx_\omega(X) = g_\omega(\nabla x(\omega), X).$$

We note that the *gradient* of x is a tangent vector $\nabla x(\omega) \in T_\omega\Omega$. Since the expression relies only on g_ω the gradient is intrinsic (assuming the existence of a Riemannian metric). We see that the gradient of a scalar field x is a vector field ∇x by considering the gradient as an operator $\nabla : \mathcal{X}(\Omega, \mathbb{R}) \to \mathcal{X}(\Omega, T\Omega)$ which takes $x(\omega) \mapsto \nabla x(\omega) \in T_\omega\Omega$.

3.4.2 Geodesics on Manifolds

Let $\gamma : [0, T] \to \Omega$ be a smooth curve on the manifold Ω with endpoints $\omega = \gamma(0)$ and $\omega' = \gamma(T)$. The derivative of the curve, known as the *velocity vector*, is a tangent vector

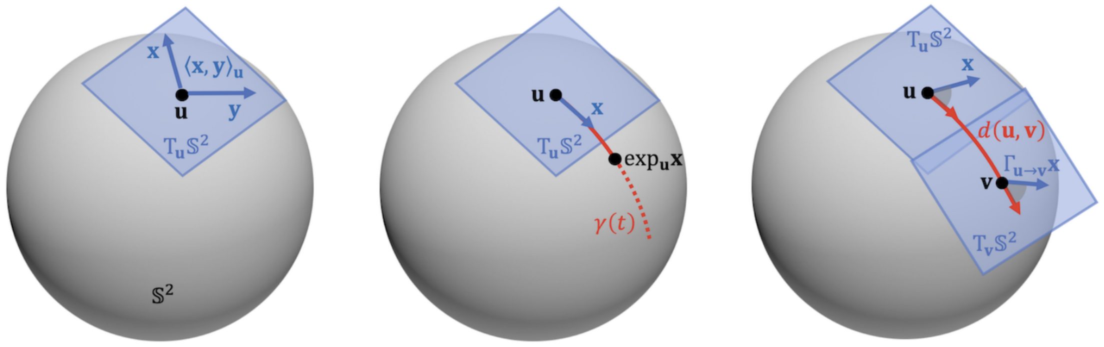


Figure 3.1: Example of Riemannian Geometry: $\Omega = S^2 = \{\boldsymbol{\omega} \in \mathbb{R}^3 : \|\boldsymbol{\omega}\| = 1\}$ is a 2-dimensional Riemannian manifold. The tangent space is a 2-dimensional plane $T_{\boldsymbol{\omega}}S^2 = \{\boldsymbol{x} \in \mathbb{R}^3 : \boldsymbol{x}^\intercal \boldsymbol{\omega} = 0\}$. The Riemannian metric is given by the Euclidean inner product: $g_{\boldsymbol{\omega}}(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\boldsymbol{\omega}} = \boldsymbol{x}^\intercal \boldsymbol{y}$ for all $\boldsymbol{x}, \boldsymbol{y} \in T_{\boldsymbol{\omega}}S^2$. The geodesics are great arcs of length $d_g(\boldsymbol{\omega}, \boldsymbol{\omega}') = \cos^{-1}(\boldsymbol{\omega}^\intercal \boldsymbol{\omega}')$. The exponential map is $\exp_{\boldsymbol{\omega}}(\boldsymbol{x}) = \cos(\|\boldsymbol{x}\|_2)\boldsymbol{\omega} + \left(\frac{\sin(\|\boldsymbol{x}\|)}{\|\boldsymbol{x}\|}\right) \boldsymbol{x}$ for $\boldsymbol{x} \in T_{\boldsymbol{\omega}}S^2$.

$\gamma'(t) \in T_{\gamma(t)}\Omega$. The curves connecting ω and ω' with *minimal length*, where the length of a curve is defined as

$$\ell(\gamma) = \int_0^T \|\gamma'(t)\|_{\gamma(t)}\, dt = \int_0^T \sqrt{g_{\gamma(t)}(\gamma'(t), \gamma'(t))}\, dt,$$

are called *geodesics* (Definition 43). Note that geodesics are intrinsic since they depend on ℓ which is a function of the Riemannian metric.

We aim to perform convolution on geodesics, which we will do via local filters in the tangent space. To do so we must first use geodesics to define *parallel transport* (a means of transporting vectors on Ω), the *exponential map* (a local intrinsic map from Ω to the tangent space), and distances via the *geodesic metric* (Definition 43). Before defining these objects we note Figure 3.1 (Page 46, Figure 11 of Bronstein et al. [1]), which showcases the basics of Riemannian geometry.

For Ω a manifold we are unable to directly add or subtract two points $\omega, \omega' \in \Omega$. We face a similar issue for tangent vectors at these two points $X \in T_\omega\Omega$ and $Y \in T_{\omega'}\Omega$ since they belong to different spaces. We can use geodesics to move these vectors between points. Let γ be a geodesic with endpoints $\omega = \gamma(0)$ and $\omega' = \gamma(T)$. Let $X \in T_\omega\Omega$ and define the tangent vectors $X(t) \in T_{\gamma(t)}\Omega$ such that the length $\ell(X(t))$ is constant and the angle between $X(t)$ and the velocity vector $\gamma'(t)$ is constant:

$$\|X(t)\|_{\gamma(t)} = \|X\|_\omega = C, \quad \langle X(t), \gamma'(t)\rangle_{\gamma(t)} = \langle X, \gamma'(0)\rangle_\omega = C'.$$

This process produces a unique vector at ω': $X(T) \in T_{\omega'}\Omega$. We describe this with the *parallel transport* map $\Gamma_{\omega\to\omega'} : T_\omega\Omega \to T_{\omega'}\Omega$ that takes $X \mapsto X(T)$. This map preserves both length and angle and so it can be thought of as a rotation of the tangent vector. We can thus associate these maps with the special orthogonal group $\mathrm{SO}(n)$ on the tangent bundle (or the orthogonal group $\mathrm{O}(n)$ for non-orientable manifolds). The map $\Gamma_{\omega \to \omega'}$ will often be represented by a group element $\mathfrak{g}_{\omega \to \omega'}$.

Given a point $\omega \in \Omega$ and a tangent vector $X \in T_\omega \Omega$ we can define a geodesic γ_X with starting point $\omega = \gamma_X(0)$ and with direction $X = \gamma'_X(0)$. If we can define $\gamma_X(t)$ for all $t \geq 0$ then we call Ω *geodesically complete* (this is always the case for compact manifolds). For such a manifold the *exponential map*, defined in the remainder of this paragraph, can be defined on the entire tangent space. The exponential map $\exp_\omega : T_\omega \Omega \to \Omega$ is defined such that $\exp_\omega(X) = \gamma_X(1)$. Let $B_r(0) \subset T_\omega \Omega$ be a ball of radius r about the origin in $T_\omega \Omega$. Note that the exponential map deforms $B_r(0)$ into a neighborhood of ω, implying that it is a local diffeomorphism. The largest r for which $\exp_\omega(B_r(0)) \subset T_\omega \Omega$ is mapped diffeomorphically is called the *injectivity radius* of M at ω. The injectivity radius of M is the infimum of the injectivity radii over all points of the manifold.

The *Hopf-Rinow theorem* tells us that geodesically complete manifolds are *complete metric spaces*, that is for any $\omega, \omega' \in \Omega$ we can define a distance known as the *geodesic distance* (Definition 43) between them using the length of a geodesic connecting them:

$$d_g(\omega, \omega') = \min\{\ell(\gamma) \text{ for } \gamma \text{ such that } \gamma(0) = \omega, \gamma(T) = \omega'\}.$$

3.4.3 Isometries and Symmetries

Let $\eta : (\Omega, g) \to (\Omega', h)$ be a diffeomorphism (Definition 38) between the Riemannian manifolds Ω and Ω' with Riemannian metrics g and h, respectively. The differential $d\eta : T\Omega \to T\Omega'$ is a map between the corresponding tangent bundles called the *pushforward*. At a given point ω the map is between tangent spaces $d\eta_\omega : T_\omega \Omega \to T_{\eta(\omega)}\Omega'$ (displacement from ω by $X \in T_\omega \Omega$ causes displacement in $\eta(\omega)$ by $d_{\eta(\omega)}(X) \in T_{\eta(\omega)}\Omega'$).

The pushforward allows us to associate the tangent vectors of the two manifolds and we can similarly *pullback* h from Ω' to Ω:

$$(\eta^* h)_\omega(X, Y) = h_{\eta(\omega)}(d\eta_\omega(X), d\eta_\omega(Y)).$$

If $g = \eta^* h$ then η is a (Riemannian) *isometry*. For 2-manifolds, isometries are deformations that do not "stretch" or "tear" the manifold.

Riemannian isometries preserve intrinsic properties such as geodesic distances implying that they are also metric isometries:

$$d_g(\omega, \omega') = d_h(\eta(\omega), \eta(\omega')), \ \forall \omega, \omega' \in \Omega.$$

Equivalently, we can write $d_g = d_h \circ (\eta \times \eta)$. On connected manifolds (an assumption we hold) the *Myers-Steenrod theorem* provides the opposite result, namely that metric isometries are

Riemannian isometries.

In the context of the blueprint, functions such as η model domain deformations. If η is an isometry then intrinsic properties of the domains are preserved. The notions of *metric dilation* and *metric distortion*, which represent the relative and absolute impacts of η on the geodesic distances, respectively, can be used to generalize exact isometries:

$$\mathrm{dil}(\eta) = \sup_{\omega,\omega' \in \Omega,\, \omega \neq \omega'} \frac{d_h(\eta(\omega), \eta(\omega'))}{d_g(\omega, \omega')}, \quad \mathrm{dis}(\eta) = \sup_{\omega,\omega' \in \Omega} |d_h(\eta(\omega), \eta(\omega')) - d_g(\omega, \omega')|.$$

Note that the *Gromov-Hausdorff distance* between metric spaces is the smallest possible metric distortion between the two spaces. Also, note that the domain deformation stability condition we previously derived for functions $f \in \mathcal{F}(\mathcal{X}(\Omega))$ can be expressed as:

$$\|f(x, \Omega) - f(x \circ \eta^{-1}, \Omega')\| \leq C\|x\|\mathrm{dis}(\eta).$$

Suppose we take $\eta \subset \mathrm{Diff}(\Omega)$ to be an automorphism on Ω (in this context a diffeomorphism from Ω to itself). Then η is a Riemannian self-isometry if $\eta^* g = g$ or a metric self-isometry if $d_g = d_g \circ (\eta \times \eta)$. Note that isometries form a group $\mathrm{Iso}(\Omega)$ called the *isometry group* with identity element given by the identity isometry that takes $\omega \mapsto \omega$. Note also that η^{-1} always exists since η is a diffeomorphism. A self-isometry of a manifold is thus an *intrinsic symmetry* of the manifold.

3.4.4 Fourier Analysis on Manifolds

Our construction of the Fourier transform in the Euclidean domain relied on the eigenvectors of circulant matrices, which by virtue of their commutativity are simultaneously diagonalizable. We can thus define the Fourier transform on general domains using any circulant matrix, in particular with a differential operator. For Riemannian geometry we will use the orthogonal eigenbasis of the *Laplacian operator*. Recall the gradient $\nabla : \mathcal{X}(\Omega, \mathbb{R}) \to \mathcal{X}(\Omega, T\Omega)$. We can similarly define the *divergence operator* $\nabla^* : \mathcal{X}(\Omega, T\Omega) \to \mathcal{X}(\Omega, \mathbb{R})$. The divergence (sometimes referred to as div) can be thought of as measuring the net flow if we think of a tangent vector field as a flow on the manifold. Note the adjointness of the two operators:

$$\langle X, \nabla x \rangle = \langle \nabla^* X, x \rangle.$$

The *Laplacian operator* on $\mathcal{X}(\Omega)$ is defined as $\Delta = \nabla^* \nabla$. We interpret the Laplacian as the difference between the average value of a function around a point (specifically around an arbitrarily small sphere around the point) and the value of said function at the point (from this interpretation we see that the Laplacian is *isotropic*, i.e. it is invariant to direction). Note that the Laplacian is self-adjoint:

$$\langle \nabla x, \nabla x \rangle = \langle x, \Delta x \rangle = \langle \Delta x, x \rangle.$$

The left-most component of the expression is really the complexity measure we saw earlier which measures the smoothness of x, which is also known as the *Dirichlet energy*:

$$c^2(x) = \|\nabla x\|^2 = \langle \nabla x, \nabla x \rangle = \int_\Omega \|\nabla x(\omega)\|_\omega^2 \, d\omega.$$

There exists an eigendecomposition for the Laplacian: $\Delta \varphi_k = \lambda_k \varphi_k$, $k \in \mathbb{N}$ with countable spectrum (Definition 51) for compact manifolds (an assumption we make for our manifolds). By self-adjointness we have orthogonality: $\langle \varphi_i, \varphi_j \rangle = \delta_{ij}$ for $\lambda_i \neq \lambda_j$. Note that we can construct this eigenbasis by iteratively minimizing the Dirichlet energy:

$$\varphi_{k+1} = \operatorname*{argmin}_{\varphi} \|\nabla \varphi\|^2 \text{ with } \|\varphi\| = 1 \text{ and } \langle \varphi, \varphi_i \rangle = 0 \text{ for } i \leq k.$$

This construction gives $0 = \lambda_0 \leq \lambda_1 \leq \lambda_2 \leq \ldots$ which leads us to think of these eigenvalues as analogous to frequencies in the standard Fourier transform.

We can now generalize the Fourier series for $x \in L^2(\Omega)$:

$$x(\omega) = \sum_{k \geq 0} \langle x, \varphi_k \rangle \varphi_k(\omega) = \sum_{k \geq 0} \hat{x}_k \varphi_k(\omega).$$

Note that the results of Aflalo and Kimmel [32, 33] prove the following optimal bound for the Laplacian eigenbasis:

$$\left\| x - \sum_{k=0}^{n} \hat{x}_k \varphi \right\|^2 \leq \frac{\|\nabla x\|^2}{\lambda_{n+1}}.$$

3.4.5 Convolution on Manifolds

We seek to construct convolution-like operations on manifolds that are intrinsic so as to be invariant to isometries. There are two methods for this: *spectral convolution* using the Fourier transform and *spatial convolution* via local matching with a filter.

3.4.5.1 Spectral Convolution

Spectral convolution between a function x and filter θ is defined using the Fourier transform and the convolution theorem:

$$(x * \theta)(\omega) = \sum_{k \geq 0} (\hat{x}_k * \hat{\theta}_k) \varphi_k(\omega).$$

Note that this definition is both isotropic (by virtue of the Laplacian being isotropic) and intrinsic.

Computing the spectral convolution, as we have defined it, requires diagonalizing the Laplacian and is thus, in general, intractable. Additionally, the high frequency eigenfunctions of the Laplacian are *geometrically unstable* (over sensitive to domain perturbations) making the spectral convolution unstable. Instead we can think of θ as a *spectral transfer function* $\hat{p}(\Delta)$ such that

$$(\hat{p}(\Delta)x)(\omega) = \sum_{k\geq 0} \hat{p}(\lambda_k)\langle x, \varphi_k\rangle\varphi_k(\omega) = \int_\Omega x(\omega')\left(\sum_{k\geq 0}\hat{p}(\lambda_k)\varphi_k(\omega)\varphi_k(\omega')\right) d\omega'.$$

Note that the sum in the above expression is a *spectral filter* with $\hat{\theta}_k = \hat{p}(\lambda_k)$ and the integral is a *spatial filter* where $\theta(\omega, \omega') = \sum_{k\geq 0}\hat{p}(\lambda_k)\varphi_k(\omega)\varphi_k(\omega')$ is a position-dependent kernel. The value added via this representation is that we can compute the spectral transfer function using a finite polynomial: $\hat{p}(\lambda) = \sum_{j=0}^{\ell}\alpha_j\lambda^j$, which allows us to avoid the spectral decomposition in computing

$$(\hat{p}(\Delta)x)(\omega) = \sum_{k\geq 0}\left(\sum_{j=0}^{\ell}\alpha_j\lambda^j\right)\langle x, \varphi_k\rangle\varphi_k(\omega) = \sum_{j=0}^{\ell}\alpha_j\cdot(\Delta^j x)(\omega).$$

3.4.5.2 Spatial Convolution

We can define spatial convolution on a manifold in a similar fashion to group convolution:

$$(x * \theta)(\omega) = \int_{T_\omega\Omega} x(\exp_\omega(X))\theta_\omega(X)\, dX.$$

Note that θ_ω is defined on the tangent space for each $\omega \in \Omega$ and thus position-dependent. If θ is intrinsic then so is the convolution defined above. We note a number of differences between manifold and group convolution:

1. Manifolds, in general, are not homogeneous spaces meaning we cannot assume a shared filter θ. Instead, we have position-dependent filters θ_ω for $\omega \in \Omega$.

2. The filter is required to be *local* and defined within the injectivity radius of Ω since the exponential map is defined locally.

3. A tangent vector $X \in T_\omega\Omega$ is a geometric abstraction so we must use a local basis $\beta_\omega : \mathbb{R}^d \to T_\omega\Omega$ to write $\theta(\boldsymbol{x}) = \theta(X)$ with coordinates $\boldsymbol{x} = \beta_\omega^{-1}(X)$.

These differences motivate us to reformulate the manifold convolution as

$$(x * \theta)(\omega) = \int_{[0,1]^n} x(\exp_\omega(\beta\boldsymbol{x}))\theta(\boldsymbol{x})\, d\boldsymbol{x}.$$

Note that the filter is defined on the unit cube in $\mathbb{R}^d$ (from locality) and that the convolution is intrinsic since the exponential map is defined via geodesics.

Consider the map $\beta_\omega : \mathbb{R}^d \to T_\omega\Omega$. We refer to this as a frame or *gauge*. We have assumed that this frame could be carried to another manifold: $\beta'_\omega = d\eta_\omega \circ \beta_\omega$. There is a difficulty here in finding such a gauge since, in general, a smooth global gauge need not exist and because there is no notion of a canonical gauge (meaning that convolutions will be different according to β). We note here that in practice the underlying manifold can be constructed to be relatively smooth and even stable so as to be (approximately) isometry-invariant. Nevertheless, despite this tension between theory and practice, we will develop the theory in our next section on the last G: gauges.

3.5 Gauges

More generally in the field of physics, a gauge is a frame for any *vector bundle* (Definition 53). A vector bundle is a collection of vector spaces V (referred to as a *fiber*) attached to each point ω in a *base space* Ω. We can think of the vector bundle as the product $\Omega \times V$ at the set level but it may have more complicated structure. In our context, each fiber of the vector bundle represents the feature space for a point on our manifold Ω. We can thus derive *gauge symmetry* from this framework. Let Ω be an n-manifold with tangent bundle $T\Omega$. Take a vector field $X : \Omega \to T\Omega$. Given a gauge β, we can represent X with a function $\boldsymbol{x} : \Omega \to \mathbb{R}^n$. This function is dependent on the gauge β, which means we must change it whenever we change the gauge so as to maintain a proper representation of X.

3.5.1 Tangent Bundles

A *gauge transformation* is a map $\mathfrak{g} : \Omega \to \mathrm{GL}(n)$ (where $\mathrm{GL}(n)$ is the *general linear group* of invertible $n \times n$ matrices) that produces a unique invertible matrix for each pair of gauges that maps between them via: $\beta'_\omega = \beta_\omega \circ \mathfrak{g}_\omega$. The corresponding coordinate representations are related via $\boldsymbol{x}'(\omega) = \mathfrak{g}_\omega^{-1}\boldsymbol{x}(\omega)$. The transformations act as representations ρ of $\mathrm{GL}(n)$. This provides the desired equality of the vector field X:

$$\begin{tikzcd}[column sep=tiny] \beta'_\omega(\boldsymbol{x}'(\omega)) \arrow[dr, equal] & = \beta_\omega(\mathfrak{g}_\omega \mathfrak{g}_\omega^{-1}\boldsymbol{x}(\omega)) = & \beta_\omega(\boldsymbol{x}(\omega)) \arrow[dl, equal] \\ & X(\omega) & \end{tikzcd}$$

We may also consider subgroups of $\mathrm{GL}(n)$ if we wish for our gauge transformations to preserve certain properties. The orthogonal group $\mathrm{O}(n)$ for example preserves orthogonality while the special orthogonal group $\mathrm{SO}(n)$ preserves orthogonality and orientation (for orientable manifolds). We call the group $\mathfrak{G}$ that represents our gauge transformations the *structure group* of the bundle. A gauge transformation then is a map from the manifold to the structure group: $\mathfrak{g} : \Omega \to \mathfrak{G}$.

Connecting back to machine learning, we can think of RGB images as tangent bundles and get an intuitive example of a gauge symmetry. An RGB image is a grid with 3 scalar values assigned to each grid representing the 3 colors. The base space then is $\Omega = \mathbb{Z}^2$ and the fibers (feature spaces) are copies of $\mathbb{R}^3$. The standard coordinate representation uses the color channels as a basis: $\boldsymbol{x}(\omega) = \begin{pmatrix} r(\omega) & g(\omega) & b(\omega) \end{pmatrix}^\mathsf{T}$. We could permute these color channels to create BRG images for example and it would not change anything about the image itself. While this operation has no practical use it demonstrates a (gauge) symmetry of the images under the (gauge) transformation of permutation that a model (function) of the images should respect. The structure group corresponding to these transformations is thus the permutation group $\mathfrak{G} = \Sigma_3$.

More concretely, if we have some model f of the images and we apply a gauge transformation to the images then we must correspondingly transform f (which equates to transforming its architecture and parameters). Suppose f maps $\boldsymbol{x}(\omega) \in \mathbb{R}^3$ to $\boldsymbol{y}(\omega) = f(\boldsymbol{x}(\omega)) \in \mathbb{R}^c$ for some c. According to the blueprint, the input and output of f are associated with group representations ρ_{in} and ρ_{out} with $\rho_{\text{in}}(\mathfrak{g}) = \mathfrak{g}$ for $\mathfrak{g} \in \mathfrak{G}$ and ρ_{out} a c-dimensional representation of $\mathfrak{G}$. If we apply a gauge transformation $\mathfrak{g}$ taking $\boldsymbol{x}(\omega)$ to $\boldsymbol{x}'(\omega)$ then we must transform f into $f' = \rho_{\text{out}}^{-1}(\mathfrak{g}) \circ f \circ \rho_{\text{in}}(\mathfrak{g})$ so that the output is $\boldsymbol{y}'(\omega) = \rho_{\text{out}}^{-1}(\mathfrak{g}_\omega)\boldsymbol{y}(\omega)$:

$$\boldsymbol{y}' = f'(\boldsymbol{x}') = \rho_{\text{out}}^{-1}(\mathfrak{g}) f(\rho_{\text{in}}(\mathfrak{g})\rho_{\text{in}}(\mathfrak{g})^{-1}\boldsymbol{x}) = \rho_{\text{out}}^{-1}(\mathfrak{g}) f(\boldsymbol{x}) = \rho_{\text{out}}^{-1}(\mathfrak{g})\boldsymbol{y}.$$

3.5.2 Gauge Symmetry

A gauge symmetry is an equivalence between gauges under a gauge transformation. Depending on our structure group, this equivalence could be between orthogonal gauges or positively-oriented orthogonal gauges, for example. Returning to the blueprint, we wish to define a function f on $\mathcal{X}(\Omega)$ that is equivariant to such a gauge transformation. Let's return to the RGB example with $f : \mathbb{R}^3 \to \mathbb{R}^c$. If $f \circ \rho_{\text{in}}(\mathfrak{g}) = \rho_{\text{out}}(\mathfrak{g}) \circ f$ then we see that f is invariant to the gauge transformation since $\rho_{\text{out}}(\mathfrak{g})^{-1} \circ f \circ \rho_{\text{in}}(\mathfrak{g}) = \rho_{\text{out}}(\mathfrak{g})^{-1} \circ \rho_{\text{out}}(\mathfrak{g}) \circ f = f$. Note that unlike our previous examples, gauge transformations act not on the domain Ω but rather on each vector $\boldsymbol{x}(\omega)$ via $\mathfrak{g}(\omega) \in \mathfrak{G}$.

We can consider more general examples. Take $f : \mathcal{X}(\Omega, \mathbb{R}) \to \mathcal{X}(\Omega, \mathbb{R})$. Note that scalar functions $x \in \mathcal{X}(\Omega, \mathbb{R})$ have no orientation so a gauge transformation has representation $\rho \equiv 1$ meaning these functions are guage-equivariant (really gauge-invariant but these are equivalent in this case). We can consider a convolution-like operation using a position-dependent filter $\theta : \Omega \times \Omega \to \mathbb{R}$:

$$(x * \theta)(\omega) = \int_\Omega \theta(\omega, \omega')x(\omega)\, d\omega'.$$

Note the existence of different filters $\theta_\omega = \theta(\omega, \cdot)$ implies the lack of spatial weight sharing.

Now take a map between vector fields $f : \mathcal{X}(\Omega, T\Omega) \to \mathcal{X}(\Omega, T\Omega)$. We realize the vector fields $X, Y \in \mathcal{X}(\Omega, T\Omega)$ (the input and output) relative to some gauge as $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}(\Omega, \mathbb{R}^n)$. We again have a position-dependent filter but it is multi-dimensional: $\boldsymbol{\Theta} : \Omega \times \Omega \to \mathbb{R}^{n \times n}$.

Note that $\mathbf{\Theta}(\omega, \omega')$ maps tangent vectors in $T_\omega\Omega$ to tangent vectors in $T_{\omega'}\Omega$. This is a problem because the different points have different gauges, which requires the overly-strong constraint: $\mathbf{\Theta}(\omega, \omega') = \rho^{-1}(\mathfrak{g}(\omega))\mathbf{\Theta}(\omega, \omega')\rho(\mathfrak{g}(\omega'))$ for all $\omega, \omega' \in \Omega$, where ρ is an $n \times n$ rotation matrix. Since we can choose $\mathfrak{g}(\omega)$ and $\mathfrak{g}(\omega')$ arbitrarily and independently our constraint ends up implying $\mathbf{\Theta} \equiv 0$.

To ameliorate this we use parallel transport $\mathfrak{g}_{\omega' \to \omega} \in \mathfrak{G}$ between the two points along the geodesic connecting them. Note that the representation of parallel transport $\rho(\mathfrak{g}_{\omega' \to \omega})$ is an $n \times n$ rotation matrix that rotates the vector during transport. We can thus define

$$(\boldsymbol{x} * \mathbf{\Theta})(\omega) = \int_\Omega \mathbf{\Theta}(\omega, \omega')\rho(\mathfrak{g}_{\omega' \to \omega})\boldsymbol{x}(\omega')\, d\omega'.$$

The transport group element under gauge transformation $\mathfrak{g}_\omega$ transforms as $\mathfrak{g}_{\omega' \to \omega} \mapsto \mathfrak{g}_{\omega'}^{-1}\mathfrak{g}_{\omega' \to \omega}\mathfrak{g}_\omega$ and the vector field transforms as $\boldsymbol{x}(\omega) \mapsto \rho(\mathfrak{g}_\omega)\boldsymbol{x}(\omega')$. If the filter $\mathbf{\Theta}$ commutes with the structure group representations: $\mathbf{\Theta}(\omega, \omega')\rho(\mathfrak{g}_\omega) = \rho(\mathfrak{g}_\omega)\mathbf{\Theta}(\omega, \omega')$ then $\boldsymbol{x} * \mathbf{\Theta}$ is a *gauge-equivariant convolution* which under gauge transformation $\boldsymbol{g}_\omega$ transforms as:

$$(\boldsymbol{x}' * \mathbf{\Theta})(\omega) = \rho^{-1}(\mathfrak{g}_\omega)(\boldsymbol{x} * \mathbf{\Theta})(\omega).$$

3.6 Meshes

Having gone through our 5 Gs we now discuss the M: *meshes*, which are somewhere in between graphs and manifolds, and *geometric graphs* which are graphs that can be geometrically realized. The additional structure on meshes allows us to, unlike standard graphs, treat them as if they were continuous. A standard approach of approximating 3-dimensional objects is with 2-manifolds. In turn, these 2-manifolds are often approximated via a discretisation into a *triangular mesh* (built by gluing triangles together along edges). See Figure 3.2 (Page 61 of Bronstein et al. [1]) for a visual.

We can define a mesh as an undirected graph with additional *triangular face structure*: $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ where we have *triangular faces* given by

$$\mathcal{F} = \{(u, v, w) : u, v, w \in \mathcal{V} \text{ and } (u, v), (u, w), (w, v) \in \mathcal{E}\}.$$

Note that the *orientation* of a face is given by the order of the vertices. We additionally assume that each edge in $\mathcal{E}$ is shared by exactly two triangles. This condition makes vertex neighborhoods disk-like rendering the mesh into a *discrete manifold* (or *manifold mesh*). If we consider the vertex features $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ we can define a metric on the mesh given by the Euclidean norm: $\ell_{uv} = \|\boldsymbol{x}_u - \boldsymbol{x}_v\|$. Analogously to the case of Riemannian manifolds, we call any property expressed solely using ℓ an intrinsic property and ℓ-preserving deformations are again isometries.

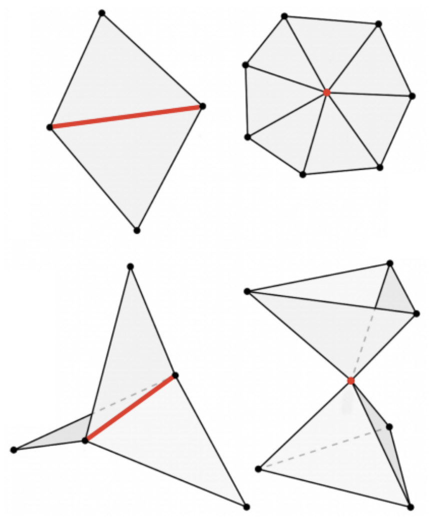


Figure 3.2: Triangular Mesh Building Blocks; Manifold (Top) and non-Manifold (Bottom)

3.6.1 Laplacian Matrices

Our vertex features represent vertex geometry as well as data-relevant properties. As we did with graphs we can stack our features into an $n \times d$ matrix $\boldsymbol{X}$. We consider spectral convolution on meshes by discretising the Laplacian operator:

$$(\boldsymbol{\Delta X})_u = \sum_{v \in \mathcal{N}_u} w_{uv}(\boldsymbol{x}_u - \boldsymbol{x}_v).$$

Alternatively, letting $d_u = \sum_v w_{uv}$ and $\boldsymbol{D} = \mathrm{diag}(d_1, d_2, ..., d_n)$ we can write $\boldsymbol{\Delta} = \boldsymbol{D} - \boldsymbol{W}$. We note that we can write $(\boldsymbol{\Delta X})_u = d_u \boldsymbol{x}_u - \sum_{v \in \mathcal{N}_u} w_{uv} \boldsymbol{x}_v$ and in the sum we see permutation-invariant aggregation of the neighbor features of u. This implies that $\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{\Delta X}$ is permutation-equivariant.

If we let $\boldsymbol{W} = \boldsymbol{A}$ where $\boldsymbol{A}$ is the adjacency matrix of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ then we see that we have a Laplacian construction for an arbitrary graph. In particular, we have not used the added mesh structure. For example, if $\mathcal{G}$ is a geometric graph with vertices having spatial coordinates then it is common to have weights inversely dependent on the metric such as $w_{uv} \propto e^{-\ell_{uv}}$.

For a mesh we can use the added structure of the faces with the *cotangent formula* [34, 35]:

$$w_{uv} = \frac{\cot \angle_{uqv} + \cot \angle_{upv}}{2a_u},$$

where the angles $\angle_{uqv}$ and $\angle_{upv}$ correspond to the angles in the triangles (u, q, v) and (u, p, v), respectively, opposite their shared edge (u, v) and where a_u is the local area computed as the area of the polygon given by the barycenters of triangles (u, p, q) containing u:

$$a_u = \frac{1}{3} \sum_{p,q:(u,p,q)\in\mathcal{F}} a_{upq}, \quad a_{upq} \text{ the area of triangle } (u, p, q).$$

See Figure 3.3 (Page 62 of Bronstein et al. [1]) for a visual of the geometry behind the cotangent formula.

Wardetzky et al. [36, 37] prove various properties of the cotangent Laplacian $\boldsymbol{\Delta}$:

1. It is positive semi-definite with eigenvalues $\lambda_n \geq ... \geq \lambda_1 \geq 0$.
2. It is symmetric implying it has orthogonal eigenvectors.
3. It is *local* in the sense that $(\boldsymbol{\Delta X})_u$ depends only on $\mathcal{N}_u$.
4. It converges to the continuous Δ for an infinitely refined (granular) mesh.

Furthermore, the cotangent Laplacian is intrinsic (giving the incredibly desirable isometry invariance property) since we can write

$$w_{uv} = \frac{-\ell_{uv}^2 + \ell_{vq}^2 + \ell_{uq}^2}{8a_{uvq}} + \frac{-\ell_{uv}^2 + \ell_{vp}^2 + \ell_{up}^2}{8a_{uvp}}$$

with a_{ijk} the area of triangle (i, j, k) given by *Heron's semiperimeter formula*:

$$a_{ijk} = \sqrt{s_{ijk}(s_{ijk} - \ell_{ij})(s_{ijk} - \ell_{ik})(s_{ijk} - \ell_{jk})}, \quad s_{ijk} = \frac{1}{2}(\ell_{ij} + \ell_{ik} + \ell_{jk}).$$

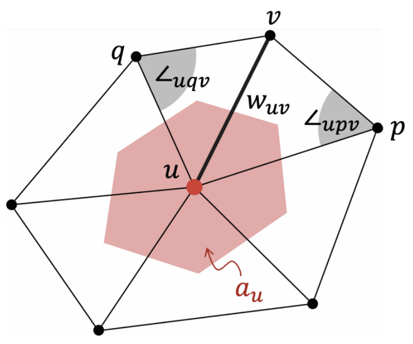


Figure 3.3: Geometry behind the Cotangent Formula

3.6.2 Spectral Convolution on Meshes

We diagonalize the Laplacian matrix as $\boldsymbol{\Delta} = \boldsymbol{\Phi\Lambda\Phi}^\mathsf{T}$ with $\boldsymbol{\Phi} = (\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2, ..., \boldsymbol{\varphi}_n)$ and $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n)$ where the λ_i's and $\boldsymbol{\varphi}_i$'s are the eigenvalues and eigenvectors of the Laplacian matrix. We can thus define spectral convolution with a filter $\boldsymbol{\theta}$ on the mesh as

$$\boldsymbol{X} * \boldsymbol{\theta} = \boldsymbol{\Phi} \cdot \text{diag}(\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\theta})(\boldsymbol{\phi}^\mathsf{T}\boldsymbol{X}) = \boldsymbol{\Phi} \cdot \text{diag}(\hat{\boldsymbol{\theta}})\hat{\boldsymbol{X}}.$$

The problem with this definition is similar to the problem with our initial definition of spectral convolution of manifolds due to the sensitivity of the Fourier transform to perturbations. We modify our definition again using transfer functions:

$$\hat{p}(\boldsymbol{\Delta})\boldsymbol{X} = \boldsymbol{\Phi} \cdot \hat{p}(\boldsymbol{\Lambda})\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{X} = \boldsymbol{\Phi} \cdot \text{diag}(\hat{p}(\lambda_1), \hat{p}(\lambda_2), ..., \hat{p}(\lambda_n))\hat{\boldsymbol{X}}.$$

We note that we can again write our transfer function as a polynomial:

$$\hat{p}(\boldsymbol{\Delta})\boldsymbol{X} = \sum_{k=0}^{\ell} \alpha_k \boldsymbol{\Delta}^k \boldsymbol{X},$$

as was done in [38].

3.6.3 Functional Maps

A *functional map* [39] is a correspondence between functions on two domains. Specifically, it is a linear operator $\boldsymbol{C} : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega')$ that can be represented by an $n' \times n$ matrix such that $\boldsymbol{x}' = \boldsymbol{C}\boldsymbol{x}$ for $\boldsymbol{x} \in \mathcal{X}(\Omega)$ and $\boldsymbol{x} \in \mathcal{X}(\Omega')$ For $\boldsymbol{C}$ to preserve area it is necessarily orthogonal [40] so we have $\boldsymbol{C} \in \text{O}(n) \implies \boldsymbol{C}^{-1} = \boldsymbol{C}^\mathsf{T}$. We will use functional maps to think of meshes as operators, allowing for invariances that exploit the mesh structure. Let $\mathcal{T}$ be a mesh with vertex coordinates $\boldsymbol{X}$. The Laplacian is an intrinsic operator representing a mesh and from which we can recover the mesh up to isometry [41]. There are other operators that can represent the mesh [42, 43, 44] and as a result we choose to represent our mesh with a general operator $\boldsymbol{Q}(\mathcal{T}, \boldsymbol{X})$, which is an $n \times n$ matrix. Instead of writing functions on the mesh as $f(\mathcal{T}, \boldsymbol{X})$ we will write $f(\boldsymbol{Q})$. See Figure 3.4 (Page 67, Figure 13 of Bronstein et al. [1]) for a visual representation of the motivation for functional maps.

Note that the vertices of a mesh are unordered, as is the case for graphs and sets. We thus are interested in permutation-invariant or permutation-equivariant functions, that is functions such that for any permutation matrix $\boldsymbol{P}$ we have:

$$f(\boldsymbol{PQP}^\mathsf{T}) = f(\boldsymbol{Q}) \text{ (invariance)}, \quad \boldsymbol{F}(\boldsymbol{PQP}^\mathsf{T}) = \boldsymbol{PF}(\boldsymbol{Q}) \text{ (equivariance)}.$$

Meshes have additional structure though given that they are discretisations of continuous surfaces. Suppose then that $\mathcal{T}$ is a discretization of a domain Ω. We could consider another mesh $\mathcal{T}'$ with n' vertices and coordinates $\boldsymbol{X}'$ that also discretizes Ω. We call this use of another mesh *remeshing*. Note that permutation does not necessarily provide a correspondence

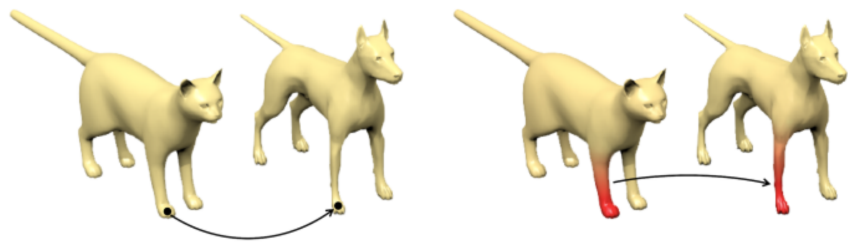


Figure 3.4: (Left) Pointwise Map of Cat/Dog Cells vs. (Right) Functional Map of Cat/Dog Shared Limb

between these meshes given that they not only have potentially distinct structure but they need not even have the same number of vertices.

This is where the functional map comes in. Suppose $\boldsymbol{Q}$ and $\boldsymbol{Q}'$ are operator representations of $\mathcal{T}$ and $\mathcal{T}'$, respectively, and that $\boldsymbol{C}$ is a correspondence (functional map) between $\mathcal{T}$ and $\mathcal{T}'$. Then we have:

$$\boldsymbol{Q}' = \boldsymbol{C}\boldsymbol{Q}\boldsymbol{C}^\top, \quad \boldsymbol{Q} = \boldsymbol{C}^\top\boldsymbol{Q}'\boldsymbol{C}.$$

This gives rise to the notions of *remeshing invariance and equivariance*, that is functions such that for any $\boldsymbol{C} \in \mathrm{O}(n)$:

$$f(\boldsymbol{Q}') = f(\boldsymbol{C}\boldsymbol{Q}\boldsymbol{C}^\top) = f(\boldsymbol{Q}) \text{ (invariance)}, \quad \boldsymbol{F}(\boldsymbol{Q}') = \boldsymbol{F}(\boldsymbol{C}\boldsymbol{Q}\boldsymbol{C}^\top) = \boldsymbol{C}\boldsymbol{F}(\boldsymbol{Q}) \text{ (equivariance)}.$$

Note that permutation is a special case of a trivial remeshing that simply reorders the unordered vertices of a mesh.

Having now explored the geometric domains we move on to applications of the theory. In the next chapter, we apply the theory and blueprint framework we have developed to different neural network architectures and in the final chapter, we will investigate a variety of practical applications of geometric deep learning.

Chapter 4

Geometric Deep Learning Models

After introducing the geometric deep learning blueprint (Definition 15) and investigating it in the context of different geometric domains, we are ready to study contemporary neural network architectures. We will aim to apply geometric deep learning to the current deep learning landscape in substantial breadth but certainly not in an exhaustive manner. That being said, we will look at three classes of models in particular, all of which we briefly discussed in Chapter 1.

The first architecture we look at is the convolutional neural network (CNN). The vanilla CNN's construction follows rather immediately from the blueprint and furthermore we will consider group-equivariant CNNs as well as CNNs on meshes.

The next architecture, whose construction also follows from the blueprint fairly easily, is the graph neural network (GNN). We will show the substantial expressive power of the GNN by introducing equivariances and we will also show that the popular transformer architecture [16] (which is the state-of-the-art in many tasks today) is in fact a GNN.

Finally, by considering a grid temporally we can study recurrent neural networks (RNNs) and show a sort-of temporal translation invariance. We will also discuss RNN pathologies that motivate models such as the Long Short-Term Memory (LSTM) [45, 46].

4.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are, unsurprisingly, reliant on convolution. Consider a finite grid (such as an image) $\Omega = [H] \times [W]$ (with $[n] = \{1, 2, ..., n\}$) and $\boldsymbol{\omega} = (\omega_1, \omega_2) \in \Omega$. We consider scalar functions $\boldsymbol{x} \in \mathcal{X}(\Omega, \mathbb{R})$. Convolving with a filter $\boldsymbol{\theta}$ of dimensions $H^f \times W^f$ is a linear combination of filter generators $\{\boldsymbol{\theta}_{ij} : i \in [H^f], j \in [W^f]\}$ (see Figure 4.1 (Page 69, Figure 14 of Bronstein et al. [1])). Now using the fact that local linear translation-equivariant maps are convolutions (from Chapter 3) we can write any local linear equivariant function in the following form:

$$\boldsymbol{F}(\boldsymbol{x}) = \sum_{i=1}^{H^f}\sum_{j=1}^{W^f} \alpha_{ij}\boldsymbol{C}(\boldsymbol{\theta}_{ij})\boldsymbol{x}.$$

If we use the standard $\boldsymbol{\theta}_{ij}(\omega_1, \omega_2) = \delta(\omega_1 - i, \omega_2 - j)$ then the above expression coincides with 2-dimensional convolution, that is:

$$\boldsymbol{F}(\boldsymbol{x})_{uv} = \sum_{i=1}^{H^f}\sum_{j=1}^{W^f} \alpha_{ij}x_{u+i,v+j}.$$

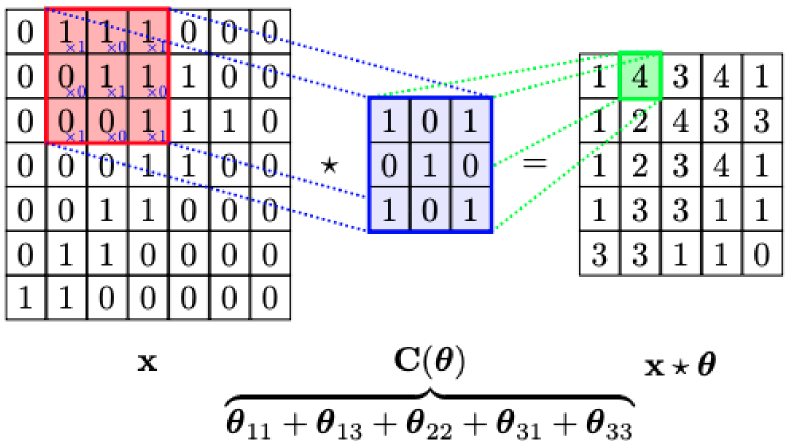


Figure 4.1: Convolution of $\boldsymbol{x}$ with Filter $\boldsymbol{C}(\boldsymbol{\theta}) = \boldsymbol{C}(\boldsymbol{\theta}_{11} + \boldsymbol{\theta}_{13} + \boldsymbol{\theta}_{22} + \boldsymbol{\theta}_{31} + \boldsymbol{\theta}_{33})$

We use a convolutional tensor for an input with M channels (for example, RGB images with $M = 3$) and an output with N channels:

$$\boldsymbol{F}(\boldsymbol{x})^{\ell}_{uv} = \sum_{i=1}^{H^f}\sum_{j=1}^{W^f}\sum_{k=1}^{M} \alpha^{\ell}_{ijk}x_{u+i,v+j,k}, \ j \in [N].$$

Between convolutional layers, a *non-linear activation function* $\sigma : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega)$ is applied element-wise $(\sigma(\boldsymbol{x}))(\omega) = \sigma(\boldsymbol{x}(\omega))$. This step is key to the model's expressive power (for neural networks in general). The most common example of such a function is the rectified linear unit or ReLU (discussed in Chapter 1): $\sigma(x) = \max(0, x)$.

Coarsening is applied after convolutional layers as well. This is done via a coarsening operator $\boldsymbol{P} : \mathcal{X}(\Omega) \to \mathcal{X}(\Omega')$ with Ω' coarser than Ω. The use of $\boldsymbol{P}$ as notation is a reference to *pooling*, which is the coarsening of choice for convolution. Pooling generally involves looking at sub-grids of the larger grid input and aggregating convolved features in said sub-grid together (via an average or a maximum, for example).

We thus see that this vanilla CNN architecture falls in line with our blueprint since we can write such a model as the composition of a coarsening operation $\boldsymbol{P}$, a non-linear activation σ, and an equivariant linear layer $\boldsymbol{F}$:

$$\boldsymbol{P}(\sigma(\boldsymbol{F}(\boldsymbol{x}))) \quad \text{(GDL blueprint expression of vanilla CNN)}$$

CNNs have had substantial impact on the field of machine learning but not without a series of innovations that made them feasible. We explore these innovations next.

A deep CNN of depth n is a model with n convolutional layers. The convolution component of such a model is specified by hyperparameters of the form $\{(H_k^f, W_k^f, M_k, N_k, p_k) : k \in [n]\}$, where the filter of the k^{th} layer is of dimension $H_K^f \times W_k^f$, M_k and N_k indicate the input and output dimensions at layer k (necessitating $M_{k+1} = N_k$), and $p_k \in \{0, 1\}$ indicates the presence of pooling. A slight modification of this architecture known as a *residual* CNN (more generally a residual network or ResNet) [47] enabled more favorable optimization of deep CNNs. This architecture relies on a skip-connection by including the input into the pooling operation:

$$\boldsymbol{P}(\boldsymbol{x} + \sigma(\boldsymbol{F}(\boldsymbol{x}))) \quad \text{(GDL blueprint expression of residual CNN).}$$

Two techniques used for CNNs but for neural networks in general are *batch normalization* [48] and *data augmentation* [49]. Batch normalization layers estimate the first and second moments of a layer $\boldsymbol{x}_k$ to estimate the layer's mean μ_k and standard deviation σ_k. These estimates are then used to normalize the layer via:

$$\boldsymbol{x}'_k = \sigma_k^{-1} \odot (\boldsymbol{x}_k - \mu_k),$$

where $\odot$ indicates element-wise multiplication.

Data augmentation involves transforming the data in a way that preserves its meaning. An example would be rotating an image or changing that image's brightness. Even adding random noise to data can be useful. This technique helps prevent overfitting and has been successful in improving model generalization. Data augmentation, however, has sub-optimal sample complexity [50] and a better approach, as we will see next, involves using invariances and equivariances.

4.1.1 Group-Equivariance

In Chapter 3, we considered the action of a group $\mathfrak{G}$ on a (homogeneous space) domain Ω and defined a corresponding group convolution. We now discuss this idea in more practical terms, first with examples of situations in which group convolutions can be useful.

We will start with discrete Ω and $\mathfrak{G}$. Let $\Omega = \mathbb{Z}^3$ be a 3-dimensional grid and take $x \in \mathcal{X}(\Omega, \mathbb{R})$ (corresponding to medical images such as MRIs perhaps). In practice, we should use a finite 3-D grid $[W] \times [H] \times [D] \subset \mathbb{Z}^3$ but in principle we can use an infinite grid $\mathbb{Z}^3$ with padding. Our symmetry group will consist of transformations on $\mathbb{Z}^3$ that preserve distance and orientation (i.e. translations and right-angle rotations): $\mathfrak{G} = \mathbb{Z}^3 \rtimes O_h$, where $\rtimes$ indicates a semi-direct product.

Another example is DNA sequences. DNA consists of sequences of four nucleotides represented by the letters A, T, C, and G. Computationally, they are represented using the *one-hot encoding* where each nucleotide is mapped to a vector e_i for $i \in [4]$ that consists of all zeroes except for a 1 at the i^{th} index. We can thus think of a DNA sequence as a function $f : \mathbb{Z} \to \mathbb{R}^4$ (really we could think of it as a map $[L] \to \mathbb{Z}^4$). DNA sequences have *reverse-complement symmetry.* They are double-stranded with a 5'-end and a 3'-end (note that DNA sequences are read from 5' to 3'). Between strands the nucleotides A and T are always paired and C and G are always paired. The sequence (5')ATGCTGCAA(3') is thus equivalent to (3')TACGACGTT(5'). Correspondingly, our symmetry group is $\mathbb{Z}_2$ with the non-identity transformation being a reverse-complementation (there is additionally translation invariance to consider since we are on a grid but this is not specific to the case of DNA). See Figure 4.2 (Page 75 of Bronstein et al. [1]) for a visual.

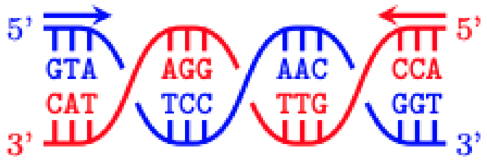


Figure 4.2: Idealized DNA Double-Helix

Since we have discrete symmetry groups, the group convolution between $x \in \mathcal{X}(\Omega, \mathbb{R})$ and a filter θ is

$$(x * \theta)(\mathfrak{g}) = \sum_{\omega \in \Omega} x_\omega \rho(\mathfrak{g}) \theta_\omega = \sum_{\omega \in \Omega} x_\omega \theta_{\mathfrak{g}^{-1}\omega}.$$

There still is the question of how to implement a group convolution in a way that is tractable and on the same (or similar) order of computation as regular convolution (and in general, deep learning methods), for which optimized hardware such as GPUs exist. Note that our discrete transformations consist of translations and another transformation (rotation for 3-D images, reverse-complementation for DNA). Thus if we take a transformation $\mathfrak{g} \in \mathfrak{G}$ we can write $\mathfrak{g} = \mathfrak{l}\mathfrak{h}$ for $\mathfrak{l} \in \mathbb{Z}^d$ a translation and $\mathfrak{g} \in \mathfrak{H}$ the other appropriate transformation. Using the fact that $\rho(\mathfrak{g}) = \rho(\mathfrak{l}\mathfrak{g}) = \rho(\mathfrak{l})\rho(\mathfrak{h})$ we can write the convolution as:

$$(x * \theta)(\mathfrak{g}) = (x * \theta)(\mathfrak{l}\mathfrak{h}) = \sum_{\omega \in \Omega} x_\omega \rho(\mathfrak{l}\mathfrak{h}) \theta_\omega$$

$$= \sum_{\omega \in \Omega} x_\omega \rho(\mathfrak{l}) \rho(\mathfrak{h}) \theta_\omega = \sum_{\omega \in \Omega} x_\omega (\rho(\mathfrak{h})\theta)_{\omega - \mathfrak{l}} = (x * (\rho(\mathfrak{h})\theta))(\mathfrak{l}).$$

The translation component of the convolution can be implemented identically to classical convolution. To complete the implementation we can pre-compute and store the transformed filters $\theta_{\mathfrak{h}} = \rho(\mathfrak{h})\theta$ for all $\mathfrak{h} \in \mathfrak{H}$ and then by accessing these transformed filters we can compute the convolution $(f * (\rho(\mathfrak{h})\theta))(\mathfrak{l}) = (f * \theta_{\mathfrak{h}})(\mathfrak{l})$ in the same way (up to accessing and storing the transformed filters) as regular convolution.

We have defined the group convolution as a function of $\mathfrak{G}$ but the fact that we can write elements of $\mathfrak{G}$ using translations and elements of $\mathfrak{H}$ motivates us to stack maps into *orientation channels*. We will have one map for each combination of filter transformation (there are $|\mathfrak{H}|$-many) and each orientation $\mathfrak{l}$. We store these maps using a $W \times H \times C$ array with C the number of channels (which is the product of the number of distinct filters and the number of non-translation filter transformations $\mathfrak{h} \in \mathfrak{H}$). The channels give an interpretation of the group-equivariance property: $(\rho(\mathfrak{g})x) * \theta = \rho(\mathfrak{g})(x * \theta)$. The group transformation on the convolution transforms the orientation channels via permutation. This gives an analogy between the group-equivariant CNN architecture and the vanilla CNN, which suggests we can extend CNN variant architectures such as residual networks to group-equivariant CNNs.

We can define CNNs on continuous domains and symmetry groups as well. Take for example *spherical CNNs* [51] defined on $\Omega = S^2$ and $\mathfrak{G} = \mathrm{SO}(3)$. We must define the Fourier transform on both spaces since convolution on S^2 is a function of SO(3). The Fourier basis on S^2 consists of *spherical harmonics* while the *Wigner D-functions* form the Fourier basis of SO(3). The Fourier transform is defined analogously to the classical transform via inner products with the Fourier basis functions and furthermore a similar result to the convolution theorem holds. Lastly, it is worth noting that there exist fast Fourier transform analogs for efficient computation of these Fourier transforms.

4.1.2 Meshes

As we have seen, meshes are very important in computer vision as they can be used to discretize 3-dimensional objects. Convolution on meshes is usually defined using the exponential map and writing the filter in coordinate form on the tangent space. Consider a geodesic γ on our domain Ω such that $\omega = \gamma(0)$ and $\omega' = \gamma(T)$ (note that on a discrete mesh a geodesic is a poly-line on the triangular faces of the mess). We can define *geodesic polar coordinates* $(r(\omega, \omega'), \vartheta(\omega, \omega'))$ where $r(\omega, \omega') = \ell(\gamma)$ is the geodesic distance between ω and ω' and $\vartheta(\omega, \omega')$ is the angle between $\gamma'(0)$ and a local reference direction. We use this polar frame to define a *geodesic patch*: $x(\omega, r, \vartheta) = x(\exp_{\omega} p_{\omega}(r, \vartheta))$ with $p_{\omega} : [0, R] \times [0, 2\pi) \to T_{\omega}\Omega$ the local polar frame (R is the injectivity radius). See Figure 4.3 (Page 86 of Bronstein et al. [1]) for a visual of geodesic patches. We now look at potential approaches to define *geodesic CNNs*.

Note that in our definition of geodesic patch we can choose orientation and direction arbitrarily and this is done in a non-invariant manner (i.e. $x(\omega, r, \vartheta + \vartheta_0)$ can differ from

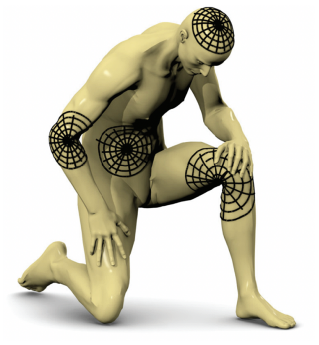


Figure 4.3: Geodesic Patches on Mesh Discretization of Human Body

$x(\omega, r, \vartheta)$). One workaround for this would be to use isotropic filters θ:

$$(x * \theta)(\omega) = \int_0^R \int_0^{2\pi} x(u, r, \vartheta)\theta(r)\; dr\; d\theta.$$

This is analogous to spectral convolution (since the Laplacian is isotropic). As a consequence of isotropicity, this definition will disregard directionality and potentially lose information.

Masci et al. [52] use a technique called *angular max pooling* to construct a geodesic CNN. This involves a non-isotropic (anisotropic) filter $\theta(r, \vartheta)$ aggregated and maximized over rotations:

$$(x * \theta)(\omega) = \max_{\vartheta_0 \in [0, 2\pi)} \int_0^R \int_0^{2\pi} x(u, r, \vartheta)\theta(r, \vartheta + \vartheta_0)\; dr\; d\vartheta.$$

We now discretize this definition using *patch operators* where for a vertex u we weigh the polar frame (r_{uv}, ϑ_{uv}) using functions on the polar coordinates $w_1, w_2, ..., w_K$ and learnable filter coefficients $\theta_1, \theta_2, ..., \theta_K$. We defer additional details to Masci et al.

The two approaches we have seen so far are *gauge-invariant* if we consider the gauge transformations $\mathfrak{g} \in \mathrm{SO}(2)$ as coordinate frame rotations. We now look at another approach [53, 54] that is *gauge-equivariant*. We must use parallel transport for our construction to map geometric features to the same vector space before convolving with the filter. We use a *message-passing* mechanism on the mesh that computes vector "messages" between each pair of mesh vertices (message-passing will be further explained in our section on graph neural networks). Take input features $\boldsymbol{x}_u \in \mathbb{R}^d$ at vertex u expressed relative to some choice of gauge at u. We assume the features transform via action by ρ_{in}, a representation of $\mathfrak{G} = \mathrm{SO}(2)$. The output $\boldsymbol{h}_u \in \mathbb{R}^{d'}$ of the convolution acts according to a representation ρ_{out}.

We define gauge-equivariant convolutions on a mesh using message-passing as:

$$\boldsymbol{h}_u = \boldsymbol{\Theta}_{\text{self}} \cdot \boldsymbol{x}_u + \sum_{v \in \mathcal{N}_u} \boldsymbol{\Theta}_{\text{neigh}}(\vartheta_{uv})\rho(\boldsymbol{g}_{v \to u})\boldsymbol{x}_v,$$

where we have learnable filters $\mathbf{\Theta}_{\text{self}}, \mathbf{\Theta}_{\text{neigh}}(\varphi_{uv}) \in \mathbb{R}^{d' \times d}$. Note that $\mathbf{\Theta}_{\text{neigh}}(\varphi_{uv})$ depends on φ_{uv}, the angle of v to the reference direction at u, which implies anisotropicity. The element $\mathfrak{g}_{v \to u} \in \text{SO}(2)$ corresponds to parallel transport from v to u (expressed relative to gauges at the vertices). The group action of this element has representation given by a *transporter matrix* $\rho(\mathfrak{g}_{v \to u}) \in \mathbb{R}^{d \times d}$. For implementation purposes, we can pre-compute these transport elements and their representations for all vertices.

We can further parameterize the filters using gauge transformation properties:

$$\boldsymbol{h}(\omega) \mapsto \rho_{\text{out}}(\mathfrak{g}^{-1}(\omega))\boldsymbol{h}(\omega) \implies \forall \vartheta \in \text{SO}(2)$$

$$\mathbf{\Theta}_{\text{self}} \cdot \rho_{\text{in}}(\vartheta) = \rho_{\text{out}}(\vartheta)\mathbf{\Theta}_{\text{self}} \quad \text{and} \quad \mathbf{\Theta}_{\text{neigh}}(\vartheta_{uv} - \vartheta)\rho_{\text{in}}(\vartheta) = \rho_{\text{out}}(\vartheta)\mathbf{\Theta}_{\text{self}}.$$

These are linear constraints implying a parameterization of the filters using basis filters and learnable coefficients of the form:

$$\mathbf{\Theta}_{\text{self}} = \sum_i \alpha_i \mathbf{\Theta}^i_{\text{self}}, \quad \mathbf{\Theta}_{\text{neigh}} = \sum_j \beta_j \mathbf{\Theta}^j_{\text{neigh}}.$$

4.2 Graph Neural Networks

Graph neural networks (GNNs) are designed to model graphical data and thus possess invariances and equivariances to permutations, but they are in fact more general and, as we will see, they can express many other neural network architectures. We model a graph as an adjacency matrix $\boldsymbol{A}$ and a matrix of node features $\boldsymbol{X}$. As per the blueprint, we will consider GNN architectures $\boldsymbol{F}(\boldsymbol{X}, \boldsymbol{A})$ that are permutation-equivariant, which we will construct using permutation-invariant functions $\phi(\boldsymbol{x}_u, \boldsymbol{X}_{\mathcal{N}_u})$ about vertices and their local neighborhoods. We can refer to the function ϕ as performing *propagation* or *message passing* and the function $\boldsymbol{F}$ as a *GNN layer*. We will now characterize three kinds of GNNs layers based on the function ϕ, which can be visualized in Figure 4.4 (Page 78, Figure 17 of Bronstein et al. [1]).

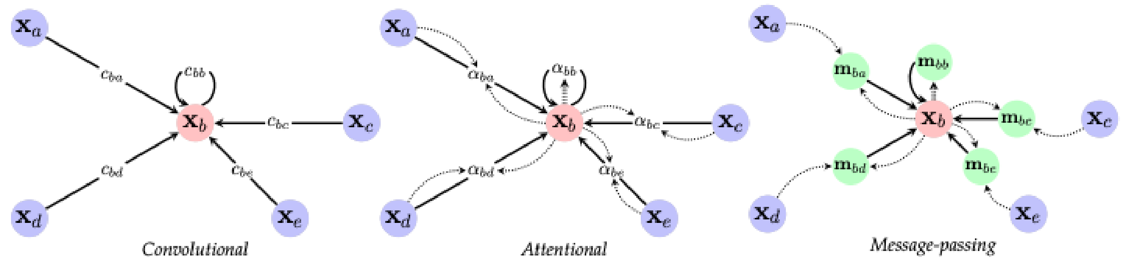


Figure 4.4: Visual Represenation of GNN Dataflows: (Left) *Convolutional GNN* with fixed weights c_{uv} between nodes (Middle) *Attentional GNN* with weights computed via attention mechanism as $\alpha_{uv} = a(\boldsymbol{x}_u, \boldsymbol{x}_v)$ between vertices (Right) *Message-Passing GNN* with messages $\boldsymbol{m}_{uv} = \psi(\boldsymbol{x}_u, \boldsymbol{x}_v)$ computed between vertices

Each different kind of GNN will satisfy permutation invariance by applying an updating function ϕ to an aggregation of vertex features from $\boldsymbol{X}_{\mathcal{N}_u}$ (transformed by some ψ) via a permutation-invariant function $\bigoplus$. The function $\bigoplus$ is generally nonparameteric such as a sum or a maximum, while ϕ and ψ are learnable. A generic example would be to use fully-connected layers with learnable weight matrices $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}$ and non-linear activation σ:

$$\psi(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \quad \phi(\boldsymbol{x}, \boldsymbol{z}) = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{U}\boldsymbol{z} + \boldsymbol{b}).$$

The first kind of GNNs are *convolutional GNNs* [55, 56, 57] which aggregate neighborhood vertices using fixed weights:

$$\boldsymbol{h}_u = \phi\left(\boldsymbol{x}_u, \bigoplus_{v \in \mathcal{N}_u} c_{uv}\psi(\boldsymbol{x}_v)\right).$$

Note that the weights c_{uv} correspond to the impact of vertex v on vertex u's representation and that we are summing over individual neighbor vertex (transformed) features $\psi(\boldsymbol{x}_v)$.

Next we have *attentional GNNs* [58, 59, 60] which use a learnable *self-attention mechanism a* used to compute weights $a(\boldsymbol{x}_u, \boldsymbol{x}_v)$:

$$\boldsymbol{h}_u = \phi\left(\boldsymbol{x}_u, \bigoplus_{v \in \mathcal{N}_u} a(\boldsymbol{x}_u, \boldsymbol{x}_v)\psi(\boldsymbol{x}_v)\right).$$

The weights α_{uv} tend to be normalized using softmax (in this case across vertex neighbors). Also, note again that we are summing over individual neighbor vertex (transformed) features.

Lastly, we have *message-passing GNNs* [10, 61] which compute arbitrary functions ("messages") across graph edges:

$$\boldsymbol{h}_u = \phi\left(\boldsymbol{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\boldsymbol{x}_u, \boldsymbol{x}_v)\right).$$

In this context, ψ is a learnable function that computes a message vector sent from vertex v to u.

We observe that convolutional GNNs can be expressed using attentional GNNs and both convolutional and attentional GNNs can be expressed using message-passing GNNs. To see this, we note that the attention mechanism $a(\boldsymbol{x}_u, \boldsymbol{x}_v) = c_{uv}$ for some fixed weights proves the initial claim and the message passing functions $\psi(\boldsymbol{x}_u, \boldsymbol{x}_v) = c_{uv}\psi(\boldsymbol{x}_v)$ (for convolutional GNNs) and $\psi(\boldsymbol{x}_u, \boldsymbol{x}_v) = a(\boldsymbol{x}_u, \boldsymbol{x}_v)\psi(\boldsymbol{x}_v)$ (for attentional GNNs) proves the latter statement. While this implies that message-passing GNNs have the most expressive power (and attentional GNNS are more powerful than convolutional GNNs) it is important to note that their power also comes with increased complexity and computational expense to train. As such, we should choose our GNN architecture according to our problem so as to not be wasteful.

4.2.1 Transformers (and more)

We now consider GNNs on unordered sets, motivated by their permutation symmetry properties. Specifically, we will show that GNNs can express the popular transformer [16] model as well as deep sets [62]. We will again consider a matrix of vertex features $\boldsymbol{X}$, but we will not assume edge structure on the graph.

We first assume an empty edge set, that is $\boldsymbol{A} = \mathbb{1}$. In this case we have trivial neighborhoods $\mathcal{N}_u = \{u\}$ for all $u \in \mathcal{V}$ (note the choice to include $u \in \mathcal{N}_u$) resulting in a GNN model of the form:

$$\boldsymbol{h}_u = \psi(\boldsymbol{x}_u),$$

with ψ a learnable function. This model is equivalent to the deep sets model.

On the flip side, we can consider the case of a complete edge set. This is useful if we do not have information regarding the graph structure and is equivalent to $\boldsymbol{A} = \mathbf{1}\mathbf{1}^\mathsf{T}$ and $\mathcal{N}_u = \mathcal{V}$ for all $u \in \mathcal{V}$. Note that convolutional GNNs in this context are of the form:

$$\boldsymbol{h}_u = \phi\left(\boldsymbol{x}_u, \bigoplus_{v \in \mathcal{V}} \psi(\boldsymbol{x}_v)\right).$$

Since the vertex aggregation term $\bigoplus_{v \in \mathcal{V}} \psi(\boldsymbol{x}_v)$ is the same for all vertices, graph convolution, in this case, is equivalent to the $\boldsymbol{A} = \mathbb{1}$ case. We instead consider an attentional GNN:

$$\boldsymbol{h}_u = \phi\left(\boldsymbol{x}_u, \bigoplus_{v \in \mathcal{V}} a(\boldsymbol{x}_u, \boldsymbol{x}_v)\psi(\boldsymbol{x}_v)\right).$$

This is the self-attention that is key to the transformer architecture. If we normalize the attention coefficients using softmax for example then we have coefficients $\alpha_{uv} = a(\boldsymbol{x}_u, \boldsymbol{x}_v)$ such that $\alpha_{uv} \in [0, 1]$, creating a *soft adjacency matrix* $\boldsymbol{\alpha}$ with rows summing to 1. We can thus model a transformer using an attentional GNN over a complete graph [63]. We only need to note that transformers model sequential data and so we must use *positional encodings* (already a component of the transformer), which are transformations of a vertex u's features $\boldsymbol{x}_u$ used to keep track of a vertex's position in the sequence. When dealing with general graph structures the graph Laplacian's eigenvectors can be used instead of positional encoding, as was done in the graph transformer [64].

We note that the empty edge set case $\boldsymbol{A} = \mathbb{1}$ is not sufficiently expressive while the complete edge set case $\boldsymbol{A} = \mathbf{1}\mathbf{1}^\mathsf{T}$ is potentially too expressive and is computationally expensive. In general, we seek to infer the edge sets of a graph, which is the challenging task of *latent graph inference*.

4.2.2 Equivariant Message-Passing

Consider the case where our vertex features include positional coordinates, an example being molecular graphs for which the vertices are atoms containing 3-dimensional spatial coordinates. Molecules are structurally equivariant to rigid transformations and thus we want our model of these molecular graphs to be SE(3)-equivariant (where SE(3) is the special Euclidean group consisting of rotations and translations).

To study this context, we will split our vertex features into (non-spatial coordinate) features $\boldsymbol{f}_u \in \mathbb{R}^d$ and *spatial coordinates* $\boldsymbol{x}_u \in \mathbb{R}^3$ and we will assume our model transforms these into $\boldsymbol{f}_u'$ and $\boldsymbol{x}_u'$, respectively. Given $\mathfrak{g} \in \mathrm{SE}(3)$ the group action is $\rho(\mathfrak{g})\boldsymbol{x} = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$ where $\boldsymbol{R} \in \mathbb{R}^{3\times 3}$ is a rotation matrix and $\boldsymbol{t} \in \mathbb{R}^3$ is a translation. We assume that the vertex features are invariant to this action (i.e. $\boldsymbol{f}_u' \mapsto \boldsymbol{f}_u'$) while the spatial coordinates are assumed to be equivariant (i.e. $\boldsymbol{x}_u' \mapsto \boldsymbol{R}\boldsymbol{x}_u' + \boldsymbol{t}$).

There is still the question of how to construct such an equivariant model. We will present a construction of E(n)-equivariant message-passing GNNs from Satorras et al. [65] (where E(n) is the Euclidean group consisting of rotations, reflections, and translations). The model acts separately on vertex features and spatial coordinates:

$$\boldsymbol{f}_u' = \phi\left(\boldsymbol{f}_u, \bigoplus_{v \in \mathcal{N}_u} \psi_f\left(\boldsymbol{f}_u, \boldsymbol{f}_v, \|\boldsymbol{x}_u - \boldsymbol{x}_v\|^2\right)\right),$$

$$\boldsymbol{x}_u' = \boldsymbol{x}_u + \sum_{v \in \mathcal{V}\setminus\{u\}} (\boldsymbol{x}_u - \boldsymbol{x}_v)\psi_x\left(\boldsymbol{f}_u, \boldsymbol{f}_v, \|\boldsymbol{x}_u - \boldsymbol{x}_v\|^2\right),$$

with distinct learnable functions ψ_f and ψ_x. The desired invariance and equivariance properties follow from construction. The only dependence of $\boldsymbol{f}_u'$ on the spatial coordinates is through the norm $\|\boldsymbol{x}_u - \boldsymbol{x}_v\|$, which is invariant to E(n), while $\boldsymbol{x}_u'$ depends on the norm $\|\boldsymbol{x}_u - \boldsymbol{x}_v\|$ (invariant) and depends linearly on the spatial coordinates, which are equivariant to E(n), implying the equivariance of $\boldsymbol{x}_u'$. Note that we have assumed the vertex features $\boldsymbol{f}_u$ are invariant to the transformation, but it might be the case that they should be equivariant (or that some of the features should be equivariant), an example being directional vectors. We can address this by separating the invariant and equivariant components and defining model updates accordingly.

4.3 Recurrent Neural Networks

We have primarily looked at domains modeling spatial data and have neglected sequential data such as time series, language, or videos. For sequential data we will consider time-step-based domains $\Omega^{(t)}$, but in general we assume $\Omega^{(t)} = \Omega$ for some domain Ω and for all t. We consider functions on these domains $\boldsymbol{X}^{(t)} \in \mathcal{X}(\Omega^{(t)})$. The inputs $\boldsymbol{X}^{(t)}$ may have non-trivial symmetries that we can exploit with geometric deep learning. For example, videos consist of frames on a fixed grid and medical scans consist of activations on a mesh that represents (parts of) the human body. More specifically, suppose we want to encode our input with a function $f(\boldsymbol{X}^{(t)})$. We consider videos as an input and so we have $\boldsymbol{X}^{(t)} \in \mathbb{R}^{n\times d}$ where n

is the number of pixels in the frame and d is the number of channels (3 channels for RBG, for example). In this case, one option is to let f be a translation-invariant CNN that at time-step t outputs $\boldsymbol{z}^{(t)} = f(\boldsymbol{X}^{(t)}) \in \mathbb{R}^k$.

We still need to aggregate information across time-steps to properly model $\boldsymbol{z}^{(t)}$. We will do this using recurrent neural networks (RNNs) and, furthermore, we will show a special temporal translation symmetry property.

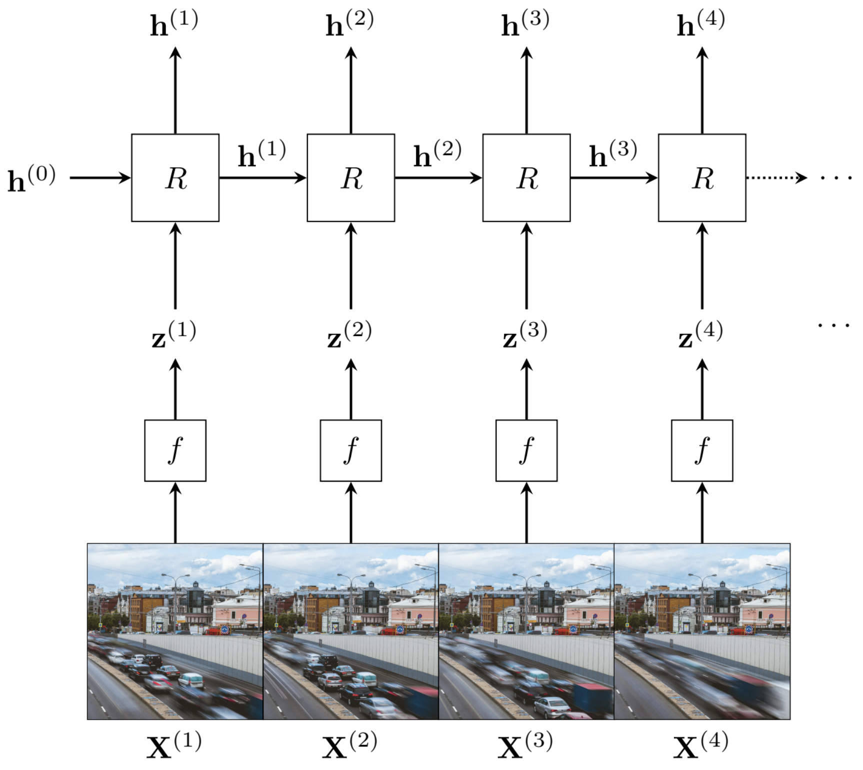


Figure 4.5: Vanilla RNN Architecture: Frames $\boldsymbol{X}^{(t)}$ Encoded to $\boldsymbol{z}^{(t)} = f(\boldsymbol{X}^{(t)})$ and Hidden States Updated via $\boldsymbol{h}^{(t)} = R(\boldsymbol{z}^{(t)}, \boldsymbol{h}^{(t-1)})$.

We presented the vanilla RNN in Chapter 1 and will do so again. See Figure 4.5 (Page 91, Figure 19 of Bronstein et al. [1]) for a visual of the architecture. This architecture computes m-dimensional hidden states $\boldsymbol{h}^{(t)} \in \mathbb{R}^m$ via an update function $R : \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}^m$:

$$\boldsymbol{h}^{(t)} = R(\boldsymbol{z}^{(t)}, \boldsymbol{h}^{(t-1)}).$$

A standard approach is to update the hidden states with a fully-connected layer, that is:

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{W}\boldsymbol{z}^{(t)} + \boldsymbol{U}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}),$$

where σ is a non-linear activation function such as ReLU and $\boldsymbol{W} \in \mathbb{R}^{k\times m}$, $\boldsymbol{U} \in \mathbb{R}^{m\times m}$, and $\boldsymbol{b} \in \mathbb{R}^m$ are learnable weights. Note that RNNs are trained using *backpropagation through* *time*, which unrolls the computation graph of the network.

The hidden states $\boldsymbol{h}^{(t)}$ are a vector representation summing the available information at time-step t. These can be used in downstream tasks for prediction. Generally, we use $\boldsymbol{h}^{(T)}$ where T is the final time-step. The initial hidden state $\boldsymbol{h}^{(0)}$ can be made learnable but it must be initialized (usually either to $\mathbf{0}$ or randomly initialized). The value of $\boldsymbol{h}^{(0)}$ is of particular interest to an RNN's translation equivariance, which we now explore.

The components of the RNN consist of discrete time-steps on a one-dimensional grid. We at first naively attempt to show an equivariance to the left-shift of the form $\boldsymbol{z}'^{(t)} = \boldsymbol{z}^{(t+1)}$. For this equivariance to hold we require $\boldsymbol{h}'^{(t)} = \boldsymbol{h}^{(t+1)}$ but this does not hold in general. Note that for $t = 1$ we have

$$\boldsymbol{h}'^{(1)} = R(\boldsymbol{z}'^{(1)}, \boldsymbol{h}^{(0)}) = R(\boldsymbol{z}^{(2)}, \boldsymbol{h}^{(0)})$$

and

$$\boldsymbol{h}^{(2)} = R(\boldsymbol{z}^{(t)}, \boldsymbol{h}^{(1)}) = R(\boldsymbol{z}^{(2)}, R(\boldsymbol{z}^{(1)}, \boldsymbol{h}^{(0)})).$$

Thus

$$\boldsymbol{h}'^{(1)} = \boldsymbol{h}^{(2)} \implies \boldsymbol{h}^{(0)} = R(\boldsymbol{z}^{(1)}, \boldsymbol{h}^{(0)}),$$

which is not true in general. Note that this is one of many constraints, the rest would be derived by looking at $t > 1$.

With the naive approach we ran into issues with boundary conditions. We can instead consider a t'-translated and left-padded state (with $t' \geq 1$):

$$\boldsymbol{z}^{*(t)} = \begin{cases} \mathbf{0} & t \leq t' \\ \boldsymbol{z}^{(t-t')} & t > t' \end{cases}.$$

Letting $\boldsymbol{z}'^{(t)} = \boldsymbol{z}^{*(t+1)}$ we have

$$\boldsymbol{h}'^{(1)} = R(\boldsymbol{z}'^{(1)}, h^{(0)}) = R(\boldsymbol{z}^{*(2)}, h^{(0)})$$

and

$$\boldsymbol{h}^{(2)} = R(\boldsymbol{z}^{*(2)}, \boldsymbol{h}^{(1)}) = R(\boldsymbol{z}^{*(1)}, \boldsymbol{h}^{(0)}) = R(\boldsymbol{z}^{*(2)}, R(\mathbf{0}, \boldsymbol{h}^{(0)})).$$

We thus have the desired equivariance: $\boldsymbol{h}'^{(t)} = \boldsymbol{h}^{(t+1)}$ if $\boldsymbol{h}^{(0)} = R(\mathbf{0}, \boldsymbol{h}^{(0)})$. This means $\boldsymbol{h}^{(0)}$ must be a fixed point of the function $\gamma(\boldsymbol{h}) = R(\mathbf{0}, \boldsymbol{h})$. For appropriately chosen R we can guarantee the existence of such a $\boldsymbol{h}^{(0)}$. Moreover, if γ is a *contraction* (i.e. $\|\gamma(\boldsymbol{x}) - \gamma(\boldsymbol{y})\| \leq r\|\boldsymbol{x} - \boldsymbol{y}\|$ with $r \in [0, 1)$) then the iteration $\boldsymbol{h}_0 = 0$, $\boldsymbol{h}_{k+1} = \gamma(\boldsymbol{h}_k)$ converges to a (unique) fixed point via the *Banach fixed point theorem.* We could continue iterating until finding n such that $\boldsymbol{h}_n = \boldsymbol{h}_{n+1}$ at which point we set $\boldsymbol{h}^{(0)} = \boldsymbol{h}_n$.

It is worth noting certain pathologies with RNNs. Because of their sequential structure they are, in a certain sense, always deep. There do exist *deep RNNs* though that consist of multiple RNN layers composed with each other. RNNs face the *vanishing gradient* and *exploding gradient* problems. What this means for the former problem is that components of the gradient that have magnitude significantly less than 1 accumulate and multiplication of these components leads to an approximately 0 (vanished) gradient. On the other hand, too large gradient components (with magnitude greater than 1) can accumulate leading to a very large (exploding) gradient. Dealing with these challenges has lead to the use of *gating mechanisms* in various RNN architectures, which we will explore next.

4.3.1 Long Short-Term Memory

RNNs today use *gating mechanisms* to maintain "memory" and "forget" things. There are two popular architectures, the *long short-term memory* or LSTM [45] and the *gated recurrent unit* (GRU) [66]. Here we will study a specific LSTM implementation [46].

LSTMs use a *memory cell* which at each time-step t computes a *cell state* $\boldsymbol{c}^{(t)} \in \mathbb{R}^m$ based on the previous cell state $\boldsymbol{c}^{(t-1)}$ as well as the previous hidden state $\boldsymbol{h}^{(t-1)}$ and the current encoded input $\boldsymbol{z}^{(t)}$. The hidden state $\boldsymbol{h}^{(t)}$ is computed using $\boldsymbol{c}^{(t)}$. To compute the cell state, we first compute *candidate features* with a fully-connected layer:

$$\boldsymbol{c}'^{(t)} = \tanh(\boldsymbol{W}_c \boldsymbol{z}^{(t)} + \boldsymbol{U}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_c),$$

where $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$ is the hyperbolic tangent activation and $\boldsymbol{W}_c$, $\boldsymbol{U}_c$, and $\boldsymbol{b}_c$ are learnable weights.

Next we compute *gating vectors* using $\boldsymbol{z}^{(t)}$ and $\boldsymbol{h}^{(t-1)}$. Specifically, we compute an *input gate* $\boldsymbol{i}^{(t)}$, *forget gate* $\boldsymbol{f}^{(t)}$, and *output gate* $\boldsymbol{o}^{(t)}$ using a fully-connected layer and the logistic sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$:

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{W}_i \boldsymbol{z}^{(t)} + \boldsymbol{U}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i) \in [0,1]^m,$$

$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{W}_f \boldsymbol{z}^{(t)} + \boldsymbol{U}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f) \in [0,1]^m,$$

and

$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{W}_o \boldsymbol{z}^{(t)} + \boldsymbol{U}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o) \in [0,1]^m,$$

where $\boldsymbol{W}_*, \boldsymbol{U}_*, \boldsymbol{b}_*$ are all learnable parameters.

Finally, we update the cell state and hidden state via using element-wise multiplication (indicated by $\odot$):

$$\boldsymbol{c}^{(t)} = \boldsymbol{i}^{(t)} \odot \boldsymbol{c}'^{(t)} + \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}^{(t)}).$$

Note that the input gate weights the candidate features, the forget gate weights the previous cell state (i.e. what to remember/forget from the past), and the output gate is used to update the hidden state. We could encode these update rules with an operator R and write

$$(\boldsymbol{h}^{(t)}, \boldsymbol{c}^{(t)}) = R(\boldsymbol{z}^{(t)}, (\boldsymbol{h}^{t-1}, \boldsymbol{c}^{(t-1)})).$$

LSTMs are effective at dealing with the vanishing gradients probably and they additionally offer *time-warping invariance* [67], which we explore now.

Suppose we are applying an RNN to a continuous $z(t)$ (analogous to our encoded input). Let $h(t)$ be a continuous analog to our hidden states $\boldsymbol{h}^{(t)}$. A first-order Taylor approximation gives:

$$h(t+\delta) \approx h(t) + \delta \cdot \frac{dh(t)}{dt}.$$

We consider $\delta = 1$ for our discrete case and note that our RNN update function R is such that

$$\frac{dh(t)}{dt} = h(t+1) - h(t) = R(z(t+1), h(t)) - h(t).$$

We want our model to be (approximately) invariant to the sampling of $z(t)$ (in the time-domain). Let $\tau : \mathbb{R}^+ \to \mathbb{R}^+$ be a *time-warping* map, that is an automorphism of time that is differentiable and monotonically increasing ($\frac{d\tau}{dt} > 0$). We say a class of models is time-warping invariant if for any model of that class and any time-warping map τ there exists another model in the class that acts on the time-warped inputs identically to how the original model acts on the un-warped inputs. The value of this invariance is that it implies that a model class is able to model long-range dependencies (which can be thought of as a time dilation). LSTMs (and more generally gated RNNs) satisfy this invariance, giving theoretical justification for their success in modeling long-range dependencies.

Time-warping by τ maps our encoded input to $z(\tau(t))$. To satisfy invariance, we want to show that the hidden state is warped to $h(\tau(t))$. We use a first-order Taylor series and the chain rule to find:

$$\frac{d(h(\tau(t)))}{d\tau(t)} = R(z(\tau(t+1)), h(\tau(t))) - h(\tau(t)) \quad \text{(Taylor expansion)}$$

$$\frac{d(h(\tau(t)))}{dt} = \frac{d(h(\tau(t)))}{d\tau(t)} \frac{d\tau(t)}{dt} = \frac{d\tau(t)}{dt} \left(R(z(\tau(t+1)), h(\tau(t))) - h(\tau(t))\right) \quad \text{(chain rule).}$$

Note that we need the derivative of the time-warping relative to normal time $\frac{d\tau(t)}{dt}$. We can estimate this with a learnable function Γ (such as a neural network): $\Gamma(z(t+1), h(t)) \approx \frac{d\tau(t)}{dt}$.

For an RNN on the time-warped domain the encoded inputs $\boldsymbol{z}^{(t)}$ and hidden states $\boldsymbol{h}^{(t)}$, correspond to $z(\tau(t))$ and $h(\tau(t))$, respectively. We again use a first-order Taylor expansion to derive the necessary condition for invariance:

$$h(\tau(t+\delta)) \approx h(\tau(t)) + \delta \cdot \frac{dh(\tau(t))}{dt}.$$

We set $\delta = 1$ and discretize to find

$$\boldsymbol{h}^{(t+1)} = \boldsymbol{h}^{(t)} + \frac{d\tau(t)}{dt} \left(R(\boldsymbol{z}^{(t+1)}, \boldsymbol{h}^{(t)}) - \boldsymbol{h}^{(t)}\right)$$

$$= \frac{d\tau(t)}{dt} R(\boldsymbol{z}^{(t+1)}, \boldsymbol{h}^{(t)}) + \left(1 - \frac{d\tau(t)}{dt}\right) \boldsymbol{h}^{(t)}$$

$$\approx \Gamma(\boldsymbol{z}^{(t+1)}, \boldsymbol{h}^{(t)}) R(\boldsymbol{z}^{(t+1)}, \boldsymbol{h}^{(t)}) + \left(1 - \Gamma(\boldsymbol{z}^{(t+1)}, \boldsymbol{h}^{(t)})\right) \boldsymbol{h}^{(t)},$$

where we substitute Γ for $\frac{d\tau(t)}{dt}$. As a sanity check, note that for vanilla RNNs the term on the right in the above expression is not present implying $\frac{d\tau(t)}{dt} = 1$ (i.e. $\tau = t$), and so vanilla RNNs are not time-warping invariant.

A few properties from our analysis motivate gated RNNs, in particular the LSTM. Note that we require $\frac{d\tau(t)}{dt} < 1$ in our expression for $\boldsymbol{h}^{(t+1)}$. This makes sense intuitively, since otherwise the warping could be "too fast" and result in us missing a data point at some time step in our discretization. We also require $\frac{d\tau(t)}{dt} > 0$ for monotonicity implying

$$\frac{d\tau(t)}{dt} \in (0,1) \implies \Gamma(\boldsymbol{z}^{(t+1)}, \boldsymbol{h}^{(t)}) \in (0,1).$$

For Γ to satisfy the above constraint we could define it using a fully-connected layer with logistic sigmoid activation σ:

$$\Gamma(\boldsymbol{z}^{(t+1)}, \boldsymbol{h}^{(t)}) = \sigma(\boldsymbol{W}_\Gamma \boldsymbol{z}^{(t+1)} + \boldsymbol{U}_\Gamma \boldsymbol{h}^{(t)} + \boldsymbol{b}_\Gamma),$$

with learnable weights $\boldsymbol{W}_\Gamma, \boldsymbol{U}_\Gamma, \boldsymbol{b}_\Gamma$. This matches the gating vector equations of the LSTM precisely. In fact, the multidimensionality of the gating vectors for an LSTM allows for time-warping invariance in multiple dimensions of $\boldsymbol{h}^{(t)}$ (with potentially different time-warpings in each dimension) [68].

Now that we have extensively developed the theory of geometric deep learning and applied it to existing neural network architectures we will conclude this thesis with a chapter on applications to real-world data and the broader field of machine learning.

Chapter 5

Applications

We conclude our foray into geometric deep learning by exploring its applications across numerous fields. Our initial discussion will regard the success of machine learning as a whole given that many of the most popular architectures, namely convolutional neural networks (CNNs), transformers, and graph neural networks (GNNs), are geometric deep learning models. We then explore practical applications in a diverse array of fields by summarizing several works in the literature and we include a high-level discussion of geometric deep learning in AlphaFold [9], the breakthrough protein structure and folding model by DeepMind. The curious reader is recommended to further investigate these works or related ones per their interest.

5.1 Machine Learning

While machine learning has impacted a massive number of fields, we will focus on computer vision (CV) and natural language processing (NLP), two fields in which it has been particularly successful in.

CNN models have revolutionized computer vision. This architecture has its humble beginnings as early as 1989 with LeNet [7] but really took the stage in 2012 with AlexNet [69], which in fact was a pivotal moment for deep learning as a whole. Since then the ResNet [47] (residual CNN) and ConvNeXt [70] architectures, among others, have been introduced. More recently, transformer architectures have been extended to images beginning with the vision transformer (ViT) [71]. ViT models have achieved state-of-the-art on many image tasks and are tied with CNNs. Some ViT architectures use convolution for enhanced results, an example being ConViT [72]. The generality of the transformer architecture is very exciting and as a result and we are currently seeing an increased popularity in *multi-modal models* such as VATT [73], which can learn using multiple kinds of data at the same time (such as images + video + audio + text).

Prior to the advent of the transformer architecture, recurrent neural networks (RNNs) were used in NLP, an example being the seq2seq (sequence-to-sequence) model [74]. These models are no longer state-of-the-art though and are being replaced by transformers. In particular, the bidirectional transformer model BERT [75] and GPT (generative pre-trained transformer) [6] have had a major impact in the field. We have already mentioned GPT-3, the state-of-the-art breaking massive 175 billion parameter language model by OpenAI, in this work but it is worth mentioning again (and certainly worth further investigation by the interested reader) given how game-changing it is.

It is important to note that CV and NLP are certainly not the only fields impacted by deep learning, but for the sake of brevity we do not delve into many of these other fields and instead opt to cover specific works that utilize geometric deep learning to a substantial extent.

5.2 Biochemistry: Molecules and Proteins

Molecules and proteins rely heavily on geometric structure and offer prominent use cases for geometric deep learning. For example, protein function is highly dependent on structure, which is usually modeled as a 3-dimensional fold. Furthermore, both of these objects can immediately be modeled as graphs with molecules, for example, consisting of atoms (vertices) and bonds between atoms (edges). This representation of molecules via molecular graphs allows for modeling them with GNNs and other geometric deep learning methods. The work by Atz et al. [76] covers the application of many of the methods we have discussed in this work, including GNNs, 3-D and Mesh CNNs, and transformers, to molecular representations.

Proteins and molecules are invariant to rotations and translations and can be invariant to reflections (when properties such as chirality are not a concern). Naturally then, we are particularly interested in model equivariance under SE(3) and E(3). Satorras et al. [65] develop E(n)-equivariant GNNs and apply them to predicting molecular properties and Batzner et al. [77] use these models for predicting interatomic potentials. Fuchs et al. develop the SE(3)-transformer [78] and also apply them to molecular property modeling. These techniques can be extended to protein-protein and protein-ligand interactions as well. For example, EQUIDOCK [79] and EQUIBIND [80] use SE(3)-equivariance for protein docking and protein binding, respectively.

For more examples of relevant works, we refer the reader to [10, 81, 82, 83, 84, 85].

5.2.1 AlphaFold

DeepMind's AlphaFold, which we have mentioned multiple times in this work already, is a hallmark work of geometric deep learning and perhaps of deep learning itself given how substantial of a scientific breakthrough it is. The model uses geometric techniques to model protein structures and is able to predict protein folds with substantial accuracy.

The AlphaFold 2 model [9] introduced multiple innovations, many of which fit in the scope of geometric deep learning. The model can be broken up into two components, the evoformer and the structure module. The evoformer is a transformer architecture modified for using evolutionary protein data. The attention mechanisms used in the evoformer are also designed for geometric protein constraints, such as the triangle inequality. The structure module incorporates SE(3)-invariance in order to make the entire model SE(3)-equivariant. This is achieved by viewing each amino acid residue as a frame represented by a tuple $T_i = (\boldsymbol{R}_i, \boldsymbol{t}_i)$ corresponding to the rotational and translational components of an element of SE(3) (i.e. $\boldsymbol{R}_i \in \mathbb{R}^{3\times3}$, $\boldsymbol{R}_i^\mathsf{T}\boldsymbol{R}_i = \mathbb{1} = \boldsymbol{R}_i\boldsymbol{R}_i^\mathsf{T}$, $\det \boldsymbol{R}_i = 1$ and $\boldsymbol{t}_i \in \mathbb{R}^3$). They use a special attention mechanism called *invariant point attention* that uses SE(3) coordinate frames to achieve the desired invariance. See Figure 5.1 ("AlphaFold" - DeepMind) for a visual of AlphaFold's success in modeling protein structure. We could certainly go into much greater detail regarding AlphaFold but for brevity we instead refer the curious reader to their paper or to DeepMind's blog post[1].

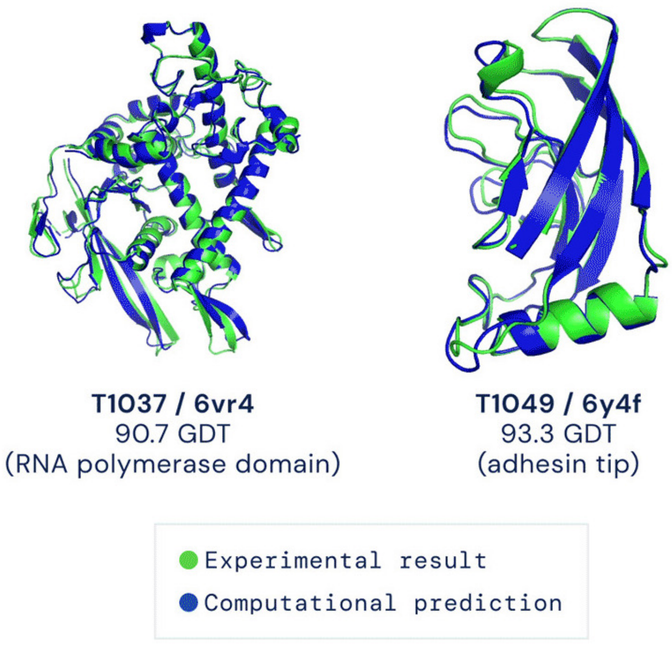


Figure 5.1: Two Target Proteins Modeled by AlphaFold - GDT Metric is Global Distance Test (Ranging from 0 to 100)

We also note that there has been great interest in the model in the literature with some noting the importance of attention and symmetries (key to geometric deep learning) in AlphaFold [15]. AlphaFold has been combined with graph transformers to predict protein-DNA binding [86]. DeepMind themselves have extended AlphaFold to predict protein multimer folds [87]. It is exciting to think about what will come next after this game-changing innovation.

[1]AlphaFold: a solution to a 50-year-old grand challenge in biology

5.3 Healthcare

Medicine offers many contexts in which geometric deep learning may be useful, including modeling parts of the human body (such as organs) as meshes and working with patient networks. CNNs, for example, can be used to diagnose diseases from retinal scans [88]. Furthermore, SE(3)-equivariant CNNs were found to outpeform vanilla CNNs on pulmonary node detection [89]. Meshes can be used to model human faces and mesh CNNs can predict faces from an individual's genetic-related demographics [90]. Furthermore, graph convolutional networks (GCNs) can be used with functional brain data (generally from an fMRI) [91] or on patient networks [92] to diagnose brain disorders such as autism. For more examples of relevant works, we refer the reader to [93, 94, 95].

5.4 Networks

Networks naturally have a graph structure suggesting that GNNs can be useful for dealing with them. Pinterest's PinSage model [96] enabled graph representation learning for graphs with millions of nodes and billions of edges. This work was followed up with graph-based recommender systems from the e-commerce companies Amazon [97] and Alibaba [98]. A similar use case is with social networks, an example being the GNN-based misinformation detection algorithm by Fabula AI [99] (acquired by Twitter).

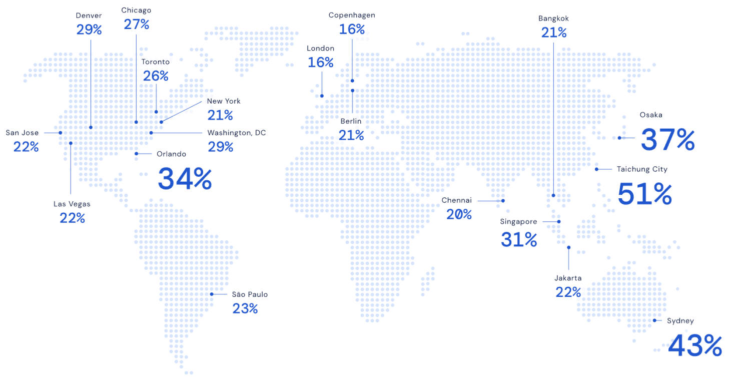


Figure 5.2: Percentage Improvements in Google Maps ETAs with GNNs

Traffic networks can also be modeled using GNNs. DeepMind used GNNs for modeling estimated times of arrival (ETAs) around the world [100]. This work has already been deployed in Google Maps. Baidu has had similar success with their ConSTGAT model [101], which is based on graph attention. See Figure 5.2 ("Traffic prediction with advanced Graph Neural Networks" - DeepMind) for a visual of %-improvements of Google Maps ETAs from using GNNs.

For more examples of relevant works, we refer the reader to [102, 103].

5.5 The Metaverse: Virtual and Augmented Reality

The term "metaverse" has been thrown around recently. It generally refers to a digital analog of what we consider the real world [104] and is being realized via virtual and augmented reality technologies. The value of geometry in this context is clear given that our world is geometric and smooth (i.e. if you take two steps to the left and turn around you do so continuously and you are still the same person). Lin et al. [105] develop *sparse steerable convolution* (SS-Conv) using sparse tensors to perform SE(3)-equivariant convolution. They apply their model to estimating object poses. The recently introduced FaceFormer [106] uses a transformer architecture to estimate facial movements according to input speech, which has the potential of making speech in the metaverse more realistic. Dundar et al. [107] use mesh learning and a custom attention mechanism to estimate 3-dimensional representations of objects from 2-D images. See Figure 5.3 (Figure 1 of Dundar et al. [107]) for examples from their work. For more examples of relevant works we refer the reader to [108, 109, 110, 111, 112].

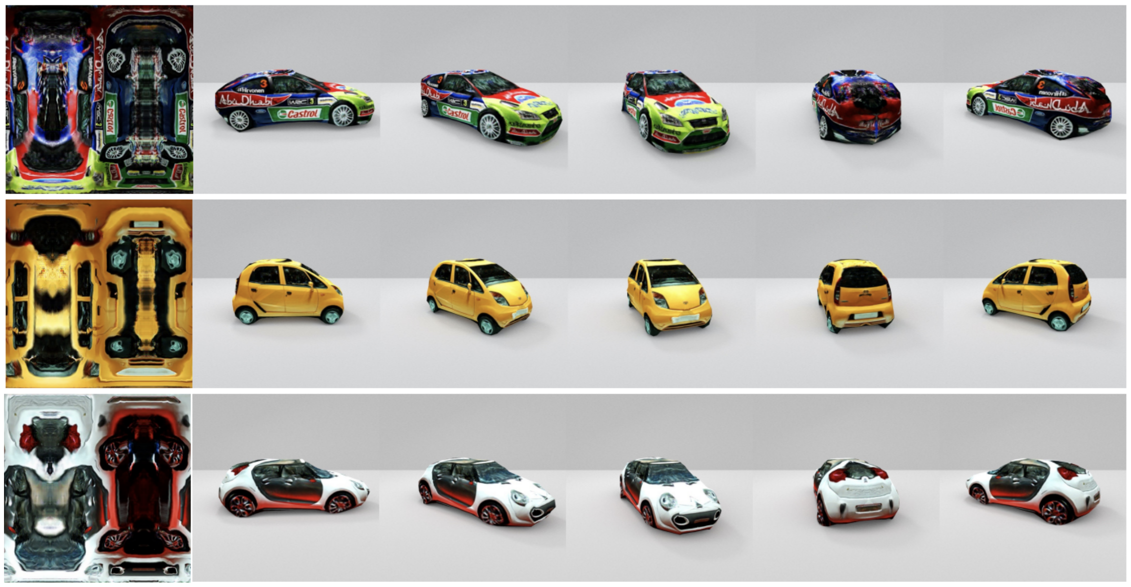


Figure 5.3: 3-D Representations of Cars from 2-D Images (using model from Dundar et al. [107])

Bibliography

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

[2] F. Klein, “A comparative review of recent researches in geometry,” *Bulletin of the American Mathematical Society*, vol. 2, pp. 215–249.

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

[4] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning.* Cambridge, England: Cambridge University Press, May 2014.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, pp. 484–489, Jan. 2016.

[9] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu,

P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, pp. 583–589, July 2021.

[10] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 1263–1272, JMLR.org, 2017.

[11] I. Cong, S. Choi, and M. D. Lukin, "Quantum convolutional neural networks," *Nature Physics*, vol. 15, pp. 1273–1278, Aug. 2019.

[12] A. Davies, P. Veličković, L. Buesing, S. Blackwell, D. Zheng, N. Tomašev, R. Tanburn, P. Battaglia, C. Blundell, A. Juhász, M. Lackenby, G. Williamson, D. Hassabis, and P. Kohli, "Advancing mathematics by guiding human intuition with AI," *Nature*, vol. 600, pp. 70–74, Dec. 2021.

[13] "Pitchbook data shows surge of ai investment in 2021 - protocol." `https://www.protocol.com/enterprise/ai-startup-funding-2021`.

[14] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. Watson, *Molecular Biology of the Cell.* Garland, 4th ed., 2002.

[15] N. Bouatta, P. Sorger, and M. AlQuraishi, "Protein structure prediction by AlphaFold2: are attention and symmetries all you need?," *Acta Crystallographica Section D Structural Biology*, vol. 77, pp. 982–991, July 2021.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[17] A. Shanehsazzadeh, D. Belanger, and D. Dohan, "Is transfer learning necessary for protein landscape prediction?," 2020.

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, p. 226–231, AAAI Press, 1996.

[19] A. M. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, pp. 433–460, 10 1950.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* The MIT Press, second ed., 2018.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *ArXiv*, vol. abs/1312.5602, 2013.

[22] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, pp. 604–609, Dec. 2020.

[23] S. Park, C. Yun, J. Lee, and J. Shin, "Minimum width for universal approximation," in *International Conference on Learning Representations*, 2021.

[24] M. Marchetti-Bowick, "Lecture notes on optimization (cmu 10-725)," Fall 2013.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[26] T. Bepler and B. Berger, "Learning protein sequence embeddings using information from structure," 2019.

[27] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2009.

[28] S. Mallat, "Group invariant scattering," *Communications on Pure and Applied Mathematics*, vol. 65, no. 10, pp. 1331–1398, 2012.

[29] B. Bamieh, "Discovering transforms: A tutorial on circulant matrices, circular convolution, and the discrete fourier transform," 2018.

[30] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[31] J. Nash, "The imbedding problem for riemannian manifolds," *Annals of Mathematics*, vol. 63, no. 1, pp. 20–63, 1956.

[32] Y. Aflalo and R. Kimmel, "Spectral multidimensional scaling," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18052–18057, 2013.

[33] Y. Aflalo, H. Brezis, and R. Kimmel, "On the optimality of shape and data representation in the spectral domain," *SIAM Journal on Imaging Sciences*, vol. 8, no. 2, pp. 1141–1160, 2015.

[34] U. Pinkall and K. Polthier, "Computing discrete minimal surfaces and their conjugates," *Experimental Mathematics*, vol. 2, no. 1, pp. 15 – 36, 1993.

[35] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, "Discrete differential-geometry operators for triangulated 2-manifolds," in *Visualization and Mathematics III*, (Berlin, Heidelberg), pp. 35–57, Springer Berlin Heidelberg, 2003.

[36] M. Wardetzky, S. Mathur, F. Kälberer, and E. Grinspun, "Discrete laplace operators: No free lunch," in *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, (Goslar, DEU), p. 33–37, Eurographics Association, 2007.

[37] M. Wardetzky, *Convergence of the Cotangent Formula: An Overview*, pp. 275–286. Basel: Birkhäuser Basel, 2008.

[38] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS'16, (Red Hook, NY, USA), p. 3844–3852, Curran Associates Inc., 2016.

[39] M. Ovsjanikov, M. Ben-Chen, J. Solomon, A. Butscher, and L. Guibas, "Functional maps: A flexible representation of maps between shapes," *ACM Trans. Graph.*, vol. 31, jul 2012.

[40] R. M. Rustamov, M. Ovsjanikov, O. Azencot, M. Ben-Chen, F. Chazal, and L. Guibas, "Map-based exploration of intrinsic shape differences and variability," *ACM Trans. Graph.*, vol. 32, jul 2013.

[41] W. Zeng, R. Guo, F. Luo, and X. Gu, "Discrete heat kernel determines discrete riemannian metric," *Graphical Models*, vol. 74, no. 4, pp. 121–129, 2012. GMP2012.

[42] D. Boscaini, D. Eynard, D. Kourounis, and M. M. Bronstein, "Shape-from-operator: Recovering shapes from intrinsic operators," *Comput. Graph. Forum*, vol. 34, p. 265–274, may 2015.

[43] E. Corman, J. Solomon, M. Ben-Chen, L. Guibas, and M. Ovsjanikov, "Functional characterization of intrinsic and extrinsic geometry," *ACM Trans. Graph.*, vol. 36, mar 2017.

[44] A. Chern, F. Knöppel, U. Pinkall, and P. Schröder, "Shape from metric," *ACM Trans. Graph.*, vol. 37, jul 2018.

[45] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, nov 1997.

[46] A. Graves, "Generating sequences with recurrent neural networks," *ArXiv*, vol. abs/1308.0850, 2013.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[48] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015.

[49] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, July 2019.

[50] S. Mei, T. Misiakiewicz, and A. Montanari, "Learning with invariances in random features and kernel models," 2021.

[51] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, "Spherical CNNs," in *International Conference on Learning Representations*, 2018.

[52] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pp. 832–840, 2015.

[53] T. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling, "Gauge equivariant convolutional networks and the icosahedral CNN," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 1321–1330, PMLR, 09–15 Jun 2019.

[54] P. D. Haan, M. Weiler, T. Cohen, and M. Welling, "Gauge equivariant mesh {cnn}s: Anisotropic convolutions on geometric graphs," in *International Conference on Learning Representations*, 2021.

[55] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[56] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, (Red Hook, NY, USA), p. 3844–3852, Curran Associates Inc., 2016.

[57] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 6861–6871, PMLR, 2019.

[58] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[59] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 5425–5434, IEEE Computer Society, jul 2017.

[60] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," in *UAI*, 2018.

[61] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Çaglar Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. M. O. Heess, D. Wierstra, P. Kohli, M. M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *ArXiv*, vol. abs/1806.01261, 2018.

[62] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[63] C. Joshi, "Transformers are graph neural networks," *The Gradient*, 2020.

[64] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *ArXiv*, vol. abs/2012.09699, 2020.

[65] V. G. Satorras, E. Hoogeboom, F. B. Fuchs, I. Posner, and M. Welling, "E(n) equivariant normalizing flows," in *Advances in Neural Information Processing Systems* (A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), 2021.

[66] K. Cho, B. van Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *EMNLP*, 2014.

[67] C. Tallec and Y. Ollivier, "Can recurrent neural networks warp time?," in *International Conference on Learning Representations*, 2018.

[68] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," *Diploma, Technische Universität München*, vol. 91, no. 1, 1991.

[69] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[70] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," 2022.

[71] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[72] S. d'Ascoli, H. Touvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun, "Convit: Improving vision transformers with soft convolutional inductive biases," in *International Conference on Machine Learning*, pp. 2286–2296, PMLR, 2021.

[73] H. Akbari, L. Yuan, R. Qian, W.-H. Chuang, S.-F. Chang, Y. Cui, and B. Gong, "VATT: Transformers for multimodal self-supervised learning from raw video, audio and text," in *Advances in Neural Information Processing Systems* (A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), 2021.

[74] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, (Cambridge, MA, USA), p. 3104–3112, MIT Press, 2014.

[75] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Association for Computational Linguistics, 2019.

[76] K. Atz, F. Grisoni, and G. Schneider, "Geometric deep learning on molecular representations," *Nature Machine Intelligence*, pp. 1–10, 2021.

[77] S. Batzner, A. Musaelian, L. Sun, M. Geiger, J. P. Mailoa, M. Kornbluth, N. Molinari, T. E. Smidt, and B. Kozinsky, "E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials," 2021.

[78] F. Fuchs, D. E. Worrall, V. Fischer, and M. Welling, "Se(3)-transformers: 3d roto-translation equivariant attention networks," in *NeurIPS*, 2020.

[79] O.-E. Ganea, X. Huang, C. Bunne, Y. Bian, R. Barzilay, T. Jaakkola, and A. Krause, "Independent se(3)-equivariant models for end-to-end rigid protein docking," 2021.

[80] H. Stärk, O.-E. Ganea, L. Pattanaik, R. Barzilay, and T. Jaakkola, "Equibind: Geometric deep learning for drug binding structure prediction," 2022.

[81] N. C. Frey, S. Samsi, J. McDonald, L. Li, C. W. Coley, and V. Gadepally, "Scalable geometric deep learning on molecular graphs," 2021.

[82] B. Anderson, T. S. Hy, and R. Kondor, "Cormorant: Covariant molecular neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[83] B. Jing, S. Eismann, P. Suriana, R. J. L. Townshend, and R. O. Dror, "Learning from protein structure with geometric vector perceptrons," *ArXiv*, vol. abs/2009.01411, 2021.

[84] B. Jing, S. Eismann, P. N. Soni, and R. O. Dror, "Equivariant graph neural networks for 3d macromolecular structure," 2021.

[85] R. J. L. Townshend, S. Eismann, A. M. Watkins, R. Rangan, M. Karelina, R. Das, and R. O. Dror, "Geometric deep learning of RNA structure," *Science*, vol. 373, pp. 1047–1051, Aug. 2021.

[86] Q. Yuan, S. Chen, J. Rao, S. Zheng, H. Zhao, and Y. Yang, "Alphafold-aware prediction of protein-dna binding sites using graph transformer," 2021.

[87] R. Evans, M. O'Neill, A. Pritzel, N. Antropova, A. Senior, T. Green, A. Žídek, R. Bates, S. Blackwell, J. Yim, O. Ronneberger, S. Bodenstein, M. Zielinski, A. Bridgland, A. Potapenko, A. Cowie, K. Tunyasuvunakool, R. Jain, E. Clancy, P. Kohli, J. Jumper, and D. Hassabis, "Protein complex prediction with AlphaFold-multimer," Oct. 2021.

[88] J. D. Fauw, J. R. Ledsam, B. Romera-Paredes, S. Nikolov, N. Tomasev, S. Blackwell, H. Askham, X. Glorot, B. O'Donoghue, D. Visentin, G. van den Driessche, B. Lakshminarayanan, C. Meyer, F. Mackinder, S. Bouton, K. Ayoub, R. Chopra, D. King, A. Karthikesalingam, C. O. Hughes, R. Raine, J. Hughes, D. A. Sim, C. Egan, A. Tufail, H. Montgomery, D. Hassabis, G. Rees, T. Back, P. T. Khaw, M. Suleyman, J. Cornebise, P. A. Keane, and O. Ronneberger, "Clinically applicable deep learning for diagnosis and referral in retinal disease," *Nature Medicine*, vol. 24, pp. 1342–1350, Aug. 2018.

[89] M. Winkels and T. S. Cohen, "Pulmonary nodule detection in ct scans with equivariant cnns," *Medical Image Analysis*, vol. 55, pp. 15–26, 2019.

[90] S. S. Mahdil, N. Nauwelaers, P. Joris, G. Bouritsas, S. Gong, S. Bokhnyak, S. Walsh, M. D. Shriver, M. Bronstein, and P. Claes, "3d facial matching by spiral convolutional metric learning and a biometric fusion-net of demographic properties," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 1757–1764, IEEE, 2021.

[91] S. I. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. J. Lee, B. Glocker, and D. Rueckert, "Distance metric learning using graph convolutional networks: Application to functional brain networks," in *MICCAI*, 2017.

[92] S. Parisot, S. I. Ktena, E. Ferrante, M. Lee, R. Guerrero, B. Glocker, and D. Rueckert, "Disease prediction using graph convolutional networks: Application to autism spectrum disorder and alzheimer's disease," *Medical Image Analysis*, vol. 48, pp. 117–130, Aug. 2018.

[93] S. Arslan, S. I. Ktena, B. Glocker, and D. Rueckert, "Graph saliency maps through spectral convolutional networks: Application to sex classification with brain connectivity," in *Graphs in biomedical image analysis and integrating medical imaging and non-imaging modalities*, pp. 3–13, Springer, 2018.

[94] B. M. Malone, A. García-Durán, and M. Niepert, "Learning representations of missing data for predicting patient outcomes," *ArXiv*, vol. abs/1811.04752, 2018.

[95] A. L. Frankel, C. Safta, C. Alleman, and R. Jones, "Mesh-based graph convolutional neural networks for modeling materials with microstructure," *Journal of Machine Learning for Modeling and Computing*, vol. 3, no. 1, 2022.

[96] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

[97] J. Hao, T. Zhao, J. Li, X. L. Dong, C. Faloutsos, Y. Sun, and W. Wang, "P-companion: A principled framework for diversified complementary product recommendation," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, (New York, NY, USA), p. 2517–2524, Association for Computing Machinery, 2020.

[98] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: A comprehensive graph neural network platform," *Proc. VLDB Endow.*, vol. 12, p. 2094–2105, aug 2019.

[99] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein, "Fake news detection on social media using geometric deep learning," *ArXiv*, vol. abs/1902.06673, 2019.

[100] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, *et al.*, "Eta prediction with graph neural networks in google maps," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3767–3776, 2021.

[101] X. Fang, J. Huang, F. Wang, L. Zeng, H. Liang, and H. Wang, "Constgat: Contextual spatial-temporal graph attention network for travel time estimation at baidu maps," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, (New York, NY, USA), p. 2697–2705, Association for Computing Machinery, 2020.

[102] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The world wide web conference*, pp. 417–426, 2019.

[103] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: A survey," 2020.

[104] N. Stephenson, *Snow Crash.* Spectra Books, May 1992.

[105] J. Lin, H. Li, K. Chen, J. Lu, and K. Jia, "Sparse steerable convolutions: An efficient learning of se (3)-equivariant features for estimation and tracking of object poses in 3d space," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[106] Y. Fan, Z. Lin, J. Saito, W. Wang, and T. Komura, "Faceformer: Speech-driven 3d facial animation with transformers," *arXiv preprint arXiv:2112.05329*, 2021.

[107] A. Dundar, J. Gao, A. Tao, and B. Catanzaro, "Fine detailed texture learning for 3d meshes with generative models," 2022.

[108] D. Kulon, R. A. Guler, I. Kokkinos, M. M. Bronstein, and S. Zafeiriou, "Weakly-supervised mesh-convolutional hand reconstruction in the wild," in *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[109] T. Shen, J. Gao, K. Yin, M.-Y. Liu, and S. Fidler, "Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis," in *Advances in Neural Information Processing Systems* (A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), 2021.

[110] B. Jiang, Y. Hong, H. Bao, and J. Zhang, "Selfrecon: Self reconstruction your digital avatar from monocular video," *arXiv preprint arXiv:2201.12792*, 2022.

[111] S. Aliakbarian, P. Cameron, F. Bogo, A. Fitzgibbon, and T. J. Cashman, "Flag: Flow-based 3d avatar generation from sparse observations," *arXiv preprint arXiv:2203.05789*, 2022.

[112] G. Daras, W.-S. Chu, A. Kumar, D. Lagun, and A. G. Dimakis, "Solving inverse problems with nerfgans," *arXiv preprint arXiv:2112.09061*, 2021.

Appendix

Here we provide supplementary mathematical background that enhances the thesis. Note that the definitions we provide are, generally speaking, more abstract than necessary.

Definition 17 (Norm)**.** A norm on a vector space X (over F a sub-field of $\mathbb{C}$) is a function $\|\cdot\| : X \to \mathbb{R}$ satisfying the following properties for all $x, y \in X$ and $c \in F$:

1. $\|cx\| = |c| \cdot \|x\|$ (absolute homogeneity)
2. $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality)
3. $\|x\| = 0 \iff x = 0$ (positive definiteness)

In general, we use $\|\cdot\|$ to denote the L^2-norm defined in Definition 18.

Definition 18 (L^p-norm)**.** Let $p \geq 1$ be a real number.

For a vector $\boldsymbol{x} \in \mathbb{R}^n$ the L^p-norm (or p-norm) of $\boldsymbol{x}$ is

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{p} x_i^p \right)^{1/p} .$$

Note that the L^2-norm (or 2-norm) is the Euclidean distance.

For a function $f : X \to \mathbb{R}$ the L^p-norm (or p-norm) of f is

$$\|f\|_p = \left(\int_X |f|^p \, dx \right)^{1/p} .$$

Definition 19 (L^p Space)**.** The L^p space on domain X, denoted as $L^p(X)$, consists of all functions $f : X \to \mathbb{R}$ with finite L^p-norm: $\|f\|_p < \infty$.

Definition 20 (Isomorphism and Automorphism)**.** Given two mathematical objects X and Y an isomorphism between X and Y is a function $f : X \to Y$ that preserves the structure of the objects and can be reversed by an inverse $f^{-1} : Y \to X$ (which also preserves structure). We write $X \cong Y$ if X and Y are isomorphic. An automorphism on X is an isomorphism between X and itself. We refer to the group of automorphisms on X as $\text{Aut}(X)$. Examples of isomorphisms include:

1. Bijections (Definition 21) on sets
2. Group isomorphisms (Definition 22)
3. Graph isomorphisms (Definition 14)
4. Homeomorphisms (Definition 30) on topological spaces
5. Bijective isometries (Definition 24) on metric spaces
6. Diffeomorphisms (Definition 38) on differential manifolds

Definition 21 (Bijection)**.** Given sets X and Y we say the map $f : X \to Y$ is bijective if it satisfies the following two properties:

1. f is one-to-one (also known as injective): $f(x_1) = f(x_2)$ if and only if $x_1 = x_2$
2. f is onto (also known as surjective): for any $y \in Y$ there exists $x \in X$ such that $f(x) = y$

Note that a bijection between X and Y admits an inverse f^{-1} and implies that the cardinalities of the sets are equal: $|X| = |Y|$. An injection of X into Y implies $|X| \preceq |Y|$ while a surjection of X onto Y implies $|X| \succeq |Y|$.

Definition 22 (Group Isomorphism)**.** Given two groups $(\mathfrak{G}, \cdot)$ and $(\mathfrak{H}, *)$ a group isomorphism between $\mathfrak{G}$ and $\mathfrak{H}$ is a bijection $f : \mathfrak{G} \to \mathfrak{H}$ that respects the group structure:

$$f(\mathfrak{g}_1 \cdot \mathfrak{g}_2) = f(\mathfrak{g}_1) * f(\mathfrak{g}_2), \ \forall \mathfrak{g}_1, \mathfrak{g}_2 \in \mathfrak{G}.$$

Definition 23 (Distance)**.** A *distance* (or *metric*) on X is a function $d : X \times X \to [0, \infty)$ that satisfies the following properties for any $x, y, z \in X$:

1. $d(x, y) = 0 \iff x = y$ (identity of indiscernibles)
2. $d(x, y) = d(y, x)$ (symmetry)
3. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Note that we can derive the non-negativity of d using the above properties. We call (X, d) a *metric space* and often omit the d when it is clear from context.

Definition 24 (Isometry)**.** Given two metric spaces (X, d_X) and (Y, d_Y) an isometry is a map $f : X \to Y$ that preserves distance:

$$d_X(x_1, x_2) = d_Y(f(x_1), f(x_2)), \ \forall x_1, x_2 \in X.$$

Definition 25 (Topological Space)**.** Given a set X and a corresponding set of subsets τ we call (X, τ) a topological space if:

1. $\emptyset \in \tau$ and $X \in \tau$
2. Any arbitrary union of elements of τ is contained in τ
3. Any finite intersection of elements of τ is contained in τ

We refer to τ as a topology on X and call the elements of τ *open sets.* A *closed set* is a subset of X whose complement is contained in τ.

Definition 26 (Neighborhoods)**.** Given a topological space X and a point $p \in X$ we call V a neighborhood of p if there exists an open set U such that $p \in U \subset V$. If U is an open set such that $p \in U \subset X$ then we call U an open neighborhood. Furthermore, if X is a metric space with distance d then the open ball of radius r about p: $B_r(p) = \{x \in X : d(x,p) < r\}$ is an open neighborhood of p. Note that we often use the term *ε-neighborhood of p* to refer to the ball $B_\varepsilon(p)$.

Definition 27 (Hausdorff)**.** A topological space X is Hausdorff if any two points in X can be separated by neighborhoods. Specifically, for any distinct $x, y \in X$ there exist disjoint open neighborhoods $U \ni x$ and $V \ni y$ (i.e. $U \cap V = \emptyset$, $U, V \subset X$). Note that most spaces we encounter will be Hausdorff.
Example: The Euclidean space $\mathbb{R}^n$ is Hausdorff.

Definition 28 (Second-Countable)**.** A topological space X is second-countable if there is a countable collection of open subsets $\mathcal{U} = \{U_i : i \in \mathbb{N}\}$ such that any open $U \subset X$ can be written as $U = \bigcup_{i=1}^{\infty} U_{n_i}$.
Example: The Euclidean space $\mathbb{R}^n$ is second-countable.

Definition 29 (Continuous Map)**.** A map $f : X \to Y$ between topological spaces is continuous if the preimage of any open set in Y is open in X:

$$C_y \text{ an open subset of } Y \implies C_x = f^{-1}(C_y) = \{x \in X : f(x) \in C_y\} \text{ an open subset of } X.$$

Definition 30 (Homeomorphism)**.** Given topological spaces X and Y we say that $f : X \to Y$ is a homeomorphism if it satisfies the following properties:

1. f is a bijection between X and Y
2. f and f^{-1} are continuous maps

Definition 31 (Manifold)**.** We require the following definition:

> **Definition 32** (Locally Euclidean)**.** We say a topological space X is locally Euclidean if there exists $n \in \mathbb{N}$ such that for every point $p \in X$ there exists a neighborhood V_p of p such that V_p is homeomorphic to $\mathbb{R}^n$.

A manifold (or topological manifold) is a locally Euclidean Hausdorff topological space. We say that M is an n-manifold if it is a manifold with neighborhoods homeomorphic to $\mathbb{R}^n$.
Examples: The Euclidean space $\mathbb{R}^n$ is an n-manifold. The n-sphere (sphere in $n+1$ dimensional real space) $S_n = \{x \in \mathbb{R}^{n+1} : \|x\| = 1\}$ is an n-manifold.

Definition 33 (Atlases and Charts)**.** For a topological space X a *chart* on X is a pair (U, φ) consisting of an open subset $U \subset X$ and φ a homeomorphism from U to an open subset of $\mathbb{R}^n$. An *atlas* for X is a collection of charts on X: $\{(U_\alpha, \varphi_\alpha) : \alpha \in I\}$ such that $X = \bigcup_{\alpha \in I} U_\alpha$ (the collection of open sets covers X). Note that a manifold can be described using an atlas where each chart describes a specific region of the manifold.

Definition 34 (Differentiable Atlas)**.** A differentiable atlas for a topological space X is a collection of charts on X such that for any two charts (U, φ) and (V, ψ) in the atlas the *transition map* $\psi \circ \varphi^{-1}$ is differentiable. Note that the transition map is a homeomorphism from $\varphi(U \cap V) \to \psi(U \cap V)$ (we assume $U \cap V \neq \emptyset$).

The motivation for this definition is the possibility of differential calculus on X. If we consider $f : X \to \mathbb{R}$ and $g : \mathbb{R} \to X$ we note that we have functions between Euclidean spaces:

$$f \circ \varphi^{-1} = (f \circ \psi^{-1}) \circ (\psi \circ \varphi^{-1}) : \mathbb{R}^n \to \mathbb{R}$$

with domain $\varphi(U \cap V)$ and

$$\varphi \circ g = (\varphi \circ \psi^{-1}) \circ (\psi \circ g) : \mathbb{R} \to \mathbb{R}^n.$$

From the decompositions above we see that for the maps $f \circ \varphi^{-1}$ and $\varphi \circ g$ to be differentiable we require that $\psi \circ \varphi^{-1}$ be differentiable, which is the definition of the differentiable atlas.

We note that differentiable can have many meanings. It could mean k-times differentiable which would give a C^k atlas. We will take it to mean infinitely differentiable or *smooth* giving a C^∞ atlas.

Lastly, the differentiable atlas admits a *maximal differentiable atlas* consisting of all charts that are *differentiably compatible* (i.e. adding the chart to the differentiable atlas produces a differentiable atlas) with said differentiable atlas.

Definition 35 (Differential Manifold)**.** A differential manifold (or differentiable manifold or smooth manifold) is a Hausdorff and second-countable topological space M with a maximal differentiable atlas on M.
Examples: The Euclidean spaces $\mathbb{R}^n$ and the n-spheres are differential manifolds.

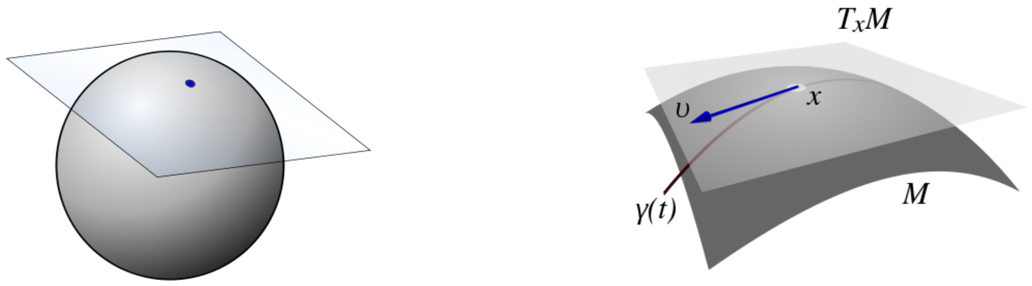
Definition 36 (Lie Group)**.** A Lie group is a group $\mathfrak{G}$ that is also a smooth manifold.

Definition 37 (Differentiable Map)**.** A continuous map between smooth manifolds $f : M \to N$ is differentiable if for any chart (U, φ) on M and any chart (V, ψ) on M the following map between (subsets) of Euclidean spaces is differentiable:

$$\psi \circ f \circ \varphi^{-1} : \varphi(U \cap f^{-1}(V)) \to \psi(V).$$

Definition 38 (Diffeomorphism)**.** A diffeomorhphism between two smooth manifolds M and N is a differential map $f : M \to N$ that is bijective and with differentiable inverse f^{-1}. We say f is a C^k-diffeomorphism if f and its inverse are k times continuously differentiable.

Definition 39 (Tangent Space)**.** Given an n-manifold M the structure constructed by attaching a tangential copy of $\mathbb{R}^n$ to each point $x \in M$ is the tangent space T_xM. A *tangent vector* is an element of the tangent space $v \in T_xM$. See Figure 5.4 ("Tangent Space" - Wikipedia) below for visuals of the tangent space.



(a) Tangent Space at a Single Point on the Sphere $S^2 = \{\boldsymbol{x} \in \mathbb{R}^3 : \|\boldsymbol{x}\|_2 = 1\}$

(b) Tangent Space T_xM of a General Manifold at $x \in M$ with Tangent Vector $v \in T_xM$ along the Curve $\gamma(t)$ through $x \in M$

Figure 5.4: Visuals of the Tangent Space

We can additionally provide a more abstract definition of the tangent space. We rely on the following definition:

Definition 40 (Equivalence Relation)**.** An equivalence relation on a set X is a binary relation $\sim$ with the following properties for all $x, y, z \in X$:

(a) $x \sim x$ (reflexivity)

(b) $x \sim y \Longleftrightarrow y \sim x$ (symmetry)

(c) $x \sim y, y \sim z \implies x \sim z$ (transitivity)

The *equivalence class* of $x \in X$ under $\sim$ is denoted as $[x] = \{y \in X : x \sim y\}$.

Suppose M is a smooth n-manifold and consider $x \in M$. Take a coordinate chart (φ, U) with $U \ni x$ an open subset of M. Take two curves $\gamma_1, \gamma_2 : (-1,1) \to M$ with start points at $x = \gamma_1(0) = \gamma_2(0)$ and suppose that $\varphi \circ \gamma_1$ and $\varphi \circ \gamma_2$, maps from $(-1,1)$ to $\mathbb{R}^n$, are differentiable. We define on equivalence relation such that the curves γ_1 and γ_2 are equivalent if the derivatives of $\varphi \circ \gamma_1$ and $\varphi \circ \gamma_2$ are equal at the point 0. The equivalence classes induced by this equivalence relation are the tangent vectors of M at x and the tangent space is the set of these equivalence classes (tangent vectors). Note that this definition is independent of the selection of chart (φ, U).

Definition 41 (Tangent Bundle)**.** For a differential manifold M the tangent bundle of M is the *disjoint union* of tangent spaces of M:

$$TM = \bigsqcup_{x \in M} T_xM.$$

Note that a disjoint union is a union in which identical objects are included as duplicates with distinct labels. See Figure 5.5 ("Tangent Bundle" - Wikipedia) below for visuals of the tangent bundle of the circle S_1.

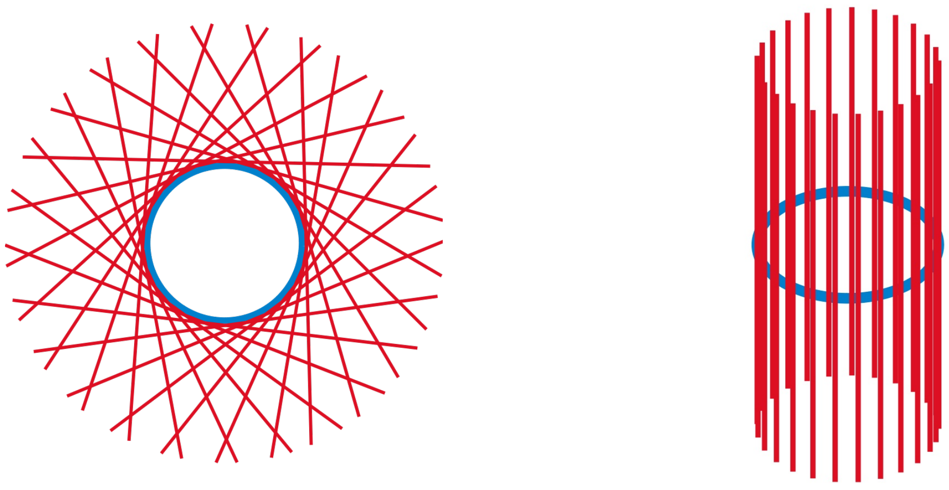


Figure 5.5: Visuals of the Tangent Bundle of the Circle $S^1 = \{\boldsymbol{x} \in \mathbb{R}^2 : \|\boldsymbol{x}\| = 1\}$: Unorganized (on left) and Organized into the Cylinder $S^1 \times \mathbb{R}$ (on right)

Definition 42 (Riemannian Manifold)**.** A Riemannian manifold is a smooth manifold M with a *Riemannian metric* g that admits a positive definite inner product g_x on the tangent space T_xM of each point $x \in M$:

$$g_x : T_xM \times T_xM \to \mathbb{R}, \quad g_x(v, v) > 0 \text{ for all } v \in T_xM \text{ such that } v \neq 0.$$

We sometimes write (M, g) to make the Riemannian metric explicit. See Figure 3.1 (Page 46, Figure 11 of Bronstein et al. [1]) for visuals of the basics of Riemannian geometry.
Examples: The Euclidean spaces $\mathbb{R}^n$ with Euclidean metric is a Riemannian manifold. The n-sphere with Euclidean distance restricted to tangent vectors is a Riemannian manifold.

Definition 43 (Geodesic)**.** Suppose (M, g) is a Riemannian manifold. For a continuously differentiable curve $\gamma : [a, b] \to M$ define the length of the curve to be

$$\ell(\gamma) = \int_a^b \sqrt{g_{\gamma(t)}(\gamma'(t), \gamma'(t))}\, dt.$$

Given points $x, y \in M$ a geodesic between x and y is a curve with start point x and end point y (i.e. $\gamma(a) = x$ and $\gamma(b) = y$) with minimal length. This length is the *geodesic distance* d_g between the points x and y:

$$d_g(x, y) = \min\{\ell(\gamma) \text{ for } \gamma \text{ such that } \gamma(a) = x, \gamma(b) = y\}.$$

See Figure 5.6 ("Deriving the Surface Area of a Spherical Triangle" - Mathematics Stack Exchange) for a visual of geodesics (specifically a spherical triangle).

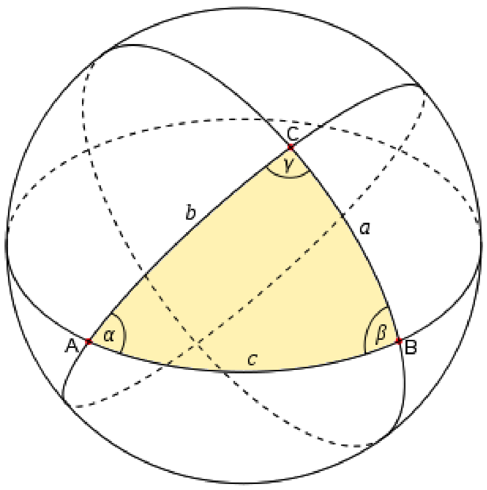


Figure 5.6: Spherical Triangle Formed by the Pairwise Intersections of Geodesics (the Three Great Arcs a, b, c) at the Three Vertices A, B, C

Definition 44 (Haar Measure)**.** The intuition behind the Haar measure is that it assigns a sort-of volume to subsets of certain kinds of groups $\mathfrak{G}$. To properly define it requires a substantial amount of mathematical machinery, which we first introduce:

Definition 45 (Topological Group)**.** A topological group is a topological space $\mathfrak{G}$ that also forms a group $(\mathfrak{G}, \cdot)$ with a continuous operator $\cdot$ and continuous group inverse map: $\mathfrak{g} \mapsto \mathfrak{g}^{-1}$.
Example: The Euclidean space $\mathbb{R}^n$ under addition is a topological group.

Definition 46 (Compact)**.** If X is a topological space then we say it is compact if for any collection of open subsets $\mathcal{U}$ such that

$$X = \bigcup_{U \in \mathcal{U}} U$$

there exists a finite sub-collection $\mathcal{F} \subset \mathcal{U}$ (called a finite cover) such that

$$X = \bigcup_{U \in \mathcal{F}} U.$$

We say $K \subset X$ is a compact subset (of X) if for any collection of open subsets $\mathcal{U}_K$ such that

$$K \subset \bigcup_{U \in \mathcal{U}_K} U$$

there exists a finite sub-collection $\mathcal{F}_K \subset \mathcal{U}_K$ (again called a finite cover) such that

$$K \subset \bigcup_{U \in \mathcal{F}_K} U.$$

Examples: The interval $[0,1]$ is a compact space. Any closed and bounded subset of $\mathbb{R}^n$ is a compact subset by the *Heine-Borel theorem*.

Definition 47 (Locally Compact)**.** We say that a topological space X is locally compact if for any point $p \in X$ there exists an open set U and a compact set K such that $x \in U \subset K$.
Examples: Compact Haussdorf spaces such as the interval $[0,1]$ are all locally compact. The space $\mathbb{R}^n$ is locally compact (again by Heine-Borel).

Definition 48 (σ-algebra)**.** Given a set X a σ-algebra (on X) is a collection of subsets $\mathcal{F}$ such that:

(a) $X \in \mathcal{F}$ and $\emptyset \in \mathcal{F}$

(b) $A \in \mathcal{F} \implies A^c = X \setminus A \in \mathcal{F}$ (closure under complementation)

(c) $A_1, A_2, \ldots \in \mathcal{F} \implies \bigcup_{n=1}^{\infty} A_n \in \mathcal{F}$ (closure under countable union)

For $\mathcal{G}$ a collection of subsets of X the σ-algebra *generated* by $\mathcal{G}$, denoted $\sigma(\mathcal{G})$, is the smallest *sigma*-algebra (minimal number of elements) that contains all elements of $\mathcal{G}$.
Example: The power set of X (the set of all subsets of X) $\mathcal{P}(X) = \{A : A \subset X\}$ is a σ-algebra.

Definition 49 (Measure)**.** Given a set X and $\mathcal{F}$ a σ-algebra on X a measure is a function $\mu : \mathcal{F} \to [0,\infty]$ such that:

(a) $\mu(\emptyset) = 0$

(b) $\mu(A) \geq 0$ for all $A \in \mathcal{F}$ (non-negativity)

(c) For $A_1, A_2, \ldots$ disjoint ($A_i \cap A_j = \emptyset$)

$$\mu\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mu(A_n) \text{ (σ-additivity or countable additivity)}$$

We say μ is non-trivial if μ is not uniquely 0. We call $(X, \mathcal{F})$ a measurable space and $(X, \mathcal{F}, \mu)$ a measure space.
Examples: The *Lebesgue measure* on $\mathbb{R}^n$ measures the n-dimensional volume (length, area, and volume in 1, 2, and 3 dimensions, respectively). If we have $\mu(X) = 1$ then $\mu = \mathbb{P}$ is a *probability measure* and $(X, \mathcal{F}, \mathbb{P})$ is a probability space.

Having defined the necessary machinery we now define the Haar measure. Suppose $\mathfrak{G}$ is a locally compact Hausdorff topological group and let $\mathcal{B}$ the Borel σ-algebra generated by all open subsets of $\mathfrak{G}$. Given $S \subset \mathfrak{G}$ we define $\mathfrak{g}S = \{\mathfrak{g}s : s \in S\}$. *Haar's theorem* implies the existence of a unique (up to multiplicative constant) nontrivial measure μ on $\mathcal{B}$ such that:

1. $\mu(\mathfrak{g}B) = \mu(B)$ for any $\mathfrak{g} \in \mathfrak{G}$ and Borel set $B \in \mathcal{B}$ (left-translation invariance)

2. $\mu(K) < \infty$ for all compact subsets $K \subset \mathfrak{G}$

3. μ is *outer regular* on Borel subsets $B \subset \mathfrak{G}$:

$$\mu(B) = \inf\{\mu(U) : B \subset U, U \text{ open}\}$$

4. μ is *inner regular* on open subsets $U \subset \mathfrak{G}$:

$$\mu(U) = \sup\{\mu(K) : K \subset U, K \text{ compact}\}$$

The above is the *left Haar measure.* A right Haar measure that is right-translation invariant (i.e. $\mu(B\mathfrak{g}) = \mu(B)$ for Borel B with $S\mathfrak{g} = \{s\mathfrak{g} : s \in S\}$) also exists. It is worth noting that the left and right Haar measures can be different.
Examples:

1. For a discrete group $\mathfrak{G}$ the Haar measure (both left and right) is the counting measure (which returns the number of elements in a subset).

2. The Haar measure for $\mathbb{R}^n$ under addition coincides with the Lebesgue measure restricted to Borel subsets of $\mathbb{R}^n$.

3. The Haar measure on the positive real numbers under multiplication is $\mu(B) = \int_B \frac{dt}{t}$ for Borel sets $B \subset \mathbb{R}$. To see the translation invariance note that for $0 < x < y$ we have $\mu((x, y)) = \log \frac{y}{x}$ and $\mu(c(x, y)) = \log \frac{cy}{cx} = \log \frac{y}{x}$.

Definition 50 (Operator)**.** The term operator is a stand-in for the term function.

Definition 51 (Spectrum)**.** The spectrum of a linear operator T (represented as a matrix) is the set of values λ such that $T - \lambda \cdot \mathbb{1}$ is not invertible. Note that any eigenvalue of T is contained in the spectrum of T.

Definition 52 (Fiber Bundle)**.** A fiber bundle is a structure (E, B, π, F) with E, B, F topological spaces (known as the *total space*, *base space*, and *fiber*, respectively) and $\pi : E \to B$ a continuous surjection called the projection map that is locally trivial in the following way: For every $x \in B$ there exists $U_x \subset B$ an open neighborhood of x and a homeomorphism $\varphi : \pi^{-1}(U_x) \to U_x \times F$ such that π coincides with $\text{proj}_{U_x}(\varphi)$ the projection of φ onto U_x. This condition is equivalent to the diagram below commuting for all $x \in B$:

$$\begin{tikzcd}
\pi^{-1}(U_x) \arrow[r, "\varphi"] \arrow[d, "\pi"] & U_x \times F \arrow[dl, "\text{proj}_{U_x}"] \\
U_x &
\end{tikzcd}$$

Note that the preimage of any point in B is homeomorphic to the fiber: $\pi^{-1}(\{x\}) \cong F$ for all $x \in B$.

Definition 53 (Vector Bundle)**.** A vector bundle is a fiber bundle (E, B, π, F) where the fiber F is a vector space.